# Capstone project – Business Analytics


Szamohval Monika Mira

ThyssenKrupp Presta AG

engineering. tomorrow. together.

thyssenkrupp

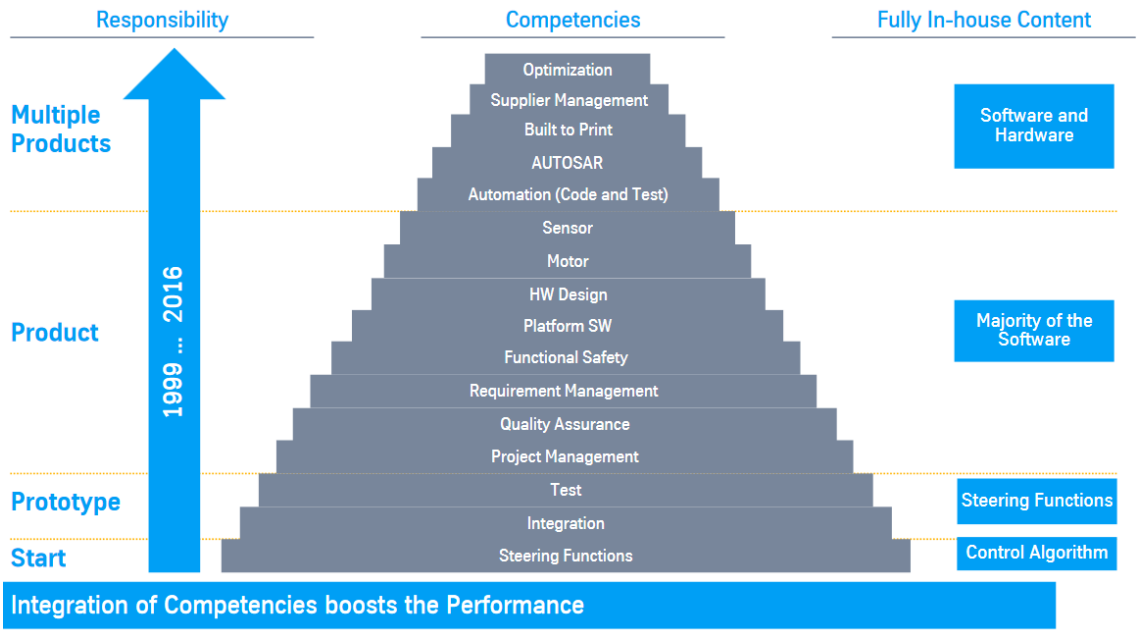
The primary goal of the following capstone project is to build new data tables from an existing data archive, that has captured years of software testing. The project objective is being tested and carried out through the use of Python programming language. The archival information and data is done through the partnership with Thyssenkrupp. In short, the project aims to help the daily work of the software testing department, by providing summary data tables comprised of years of test archives from various software testing projects.

## The company

Thyssenkrupp Presta, located in Budapest, is one of the leading automotive industry and component suppliers worldwide. Nine-of-ten premium cars currently on the market contain their components and one-of-every-three trucks contain a Thyssenkrupp propulsion. The Budapest group, Thyssenkrupp Presta Hungary Kft., belongs to the Component Technology portion of the business. The Hungarian unit, is a Competence centre with more than 500 employees. Currently the centre operates in the field of steering technology research and development. Also, the company operates an automotive factory in Győr, which supplies complete chassis as a direct supplier to car factories, into so called just-in-sequence systems. Moreover, the company opened a new factory in Jászfényszaru, producing steering systems (C-EPS) and cylinder head integrated camshafts, with the help of about 500 staff members. In addition to its automotive industry, Thyssenkrupp employs around 400 people in the elevator technology, and in industrial materials and related services. Through the years the company reached multiple products level from prototypes. The competency level growth is outstanding.



CEU Department of Economics and Business
CENTRAL EUROPEAN UNIVERSITY

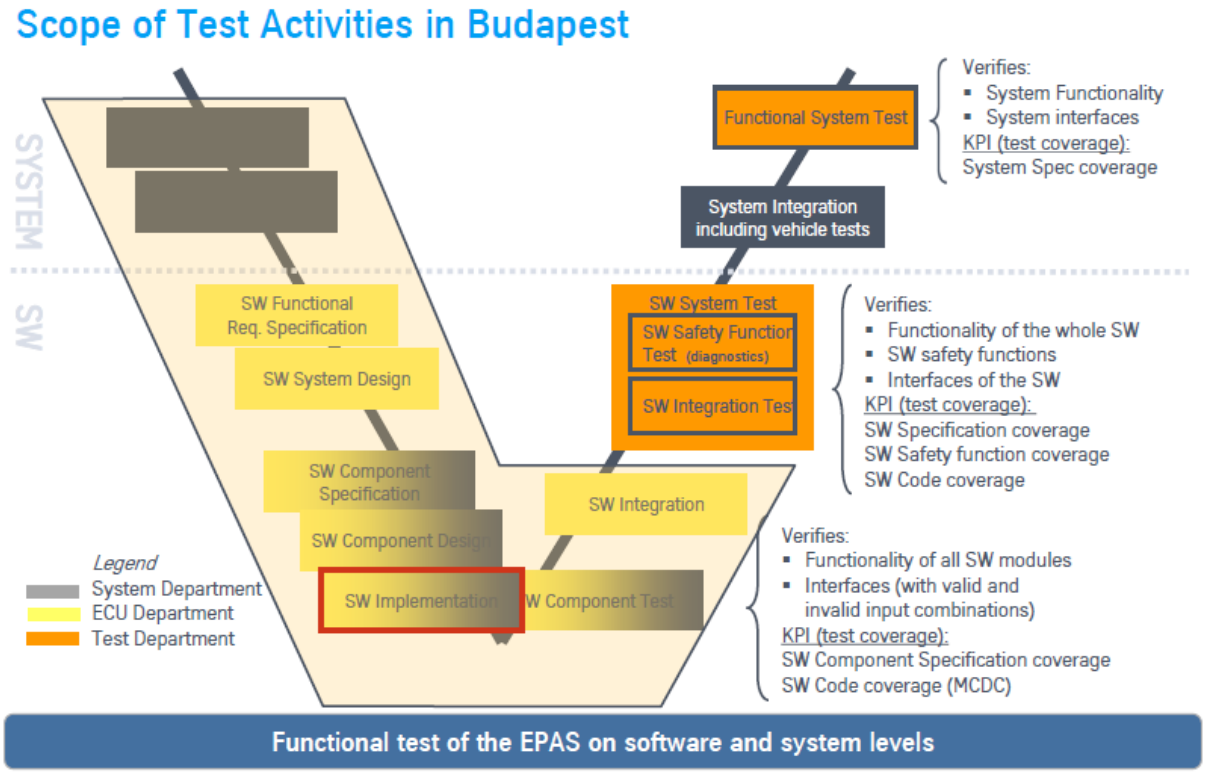



The chart above clearly illustrates the development lifecycle of the company through the years.

## The testing team

In the following chapter I am going to detail the Budapest team, as my capstone project was fulfilled at the competence centre, in Presta Kft., Budapest. The Hungarian testing team, is a group of engineers, and their focus is developing and testing electromechanical software-controlled steering systems into self-propelled cars which are working with visual signal processing. The team develops and tests smart steering systems which utilize electromotive aid, instead of the traditional which uses a continuously energizing hydraulic system. The electromotive system only consumes energy when the steering wheel is rotated and therefore saves up to 0.3 litres of fuel per hundred kilometres depending on the car type, also it is technically more reliable than the traditional fuel regeneration. It is essential for self- guided cars, as the software control provides, a whole range of security and convenience features, such as side wind compensation or a parking assistant, in addition to it, it can provide crucial access to the steering systems of the self-governing cars. The team originally started testing the software of the self-governing cars, because the competence centre in Budapest had begun developing steering systems into it. Despite this, these systems will only be put into street cars in the following 10 to 15 years. With such advantage in time, the Hungarian System is years ahead of their competitors. Their work is unquestionably important, as the emergence of the self-guided cars will create a huge paradigm shift in the world; there still remains the question as to what conditions will allow for drive automatic systems on be fully functioning on the road. Yet what we know now is

that electronic systems are *always paying attention, never get tired, and their reaction times already outperform humans*.

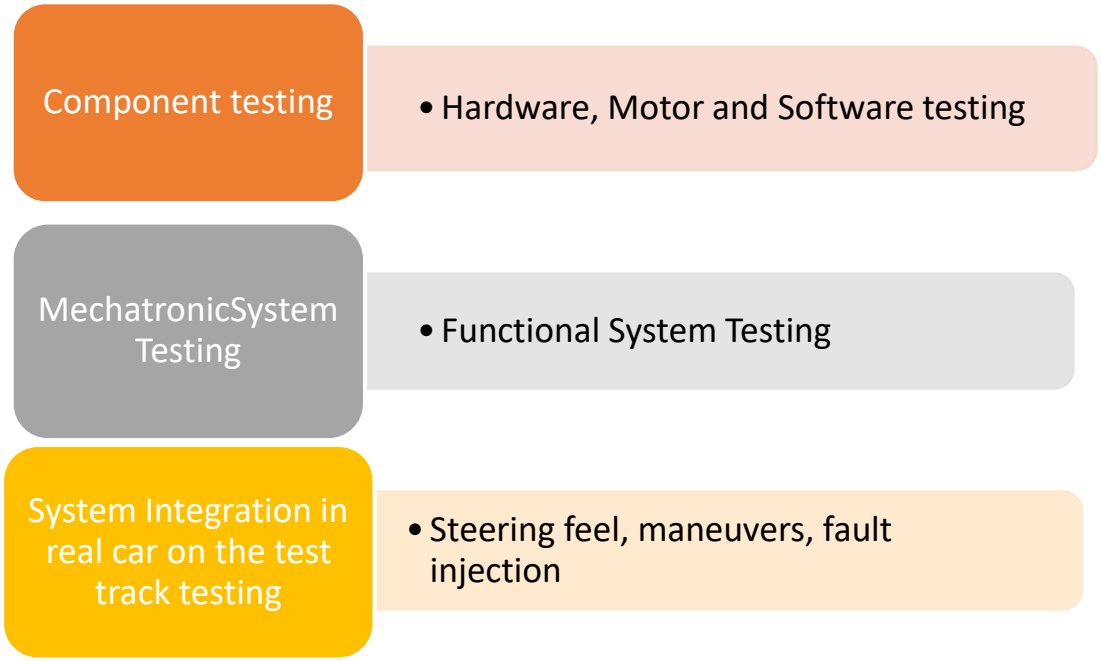



## The method of testing

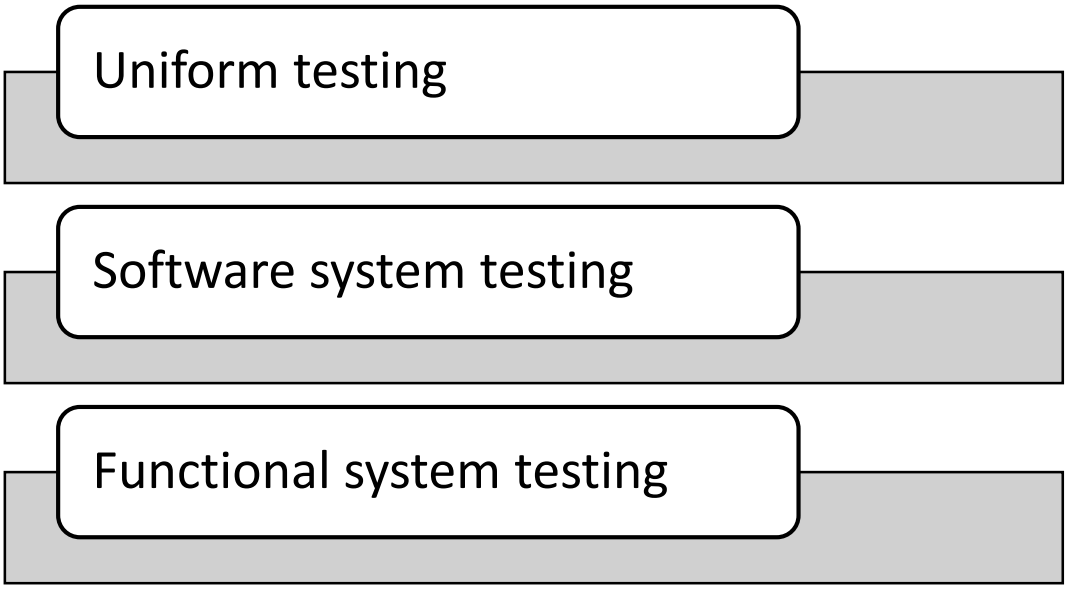
Why is software testing needed? First, to prove that the system works. In addition, it finds as many faults as possible, as soon as possible, during the development lifecycle. Moreover, it demonstrates conformance to requirements both to customers and towards to the internal requirements as well. Testing is required as engineers need to find the errors in the software products, before placing them into operation, thus increasing the quality and reliability of the product. Almost, we can be sure that there is a bug in the software, as people develop them, and they make mistakes. After testing, it can be stated that there is no error in the tested parts, so the reliability of the product increases. The reality is that testing cannot prove that the system is correct! Testing can show what the system can and cannot do. Testing cannot in fact prove that the system is correct. Further, testing is required to understand the system and deep dive in the details of the system, and get know as many details as possible, with discovering all the side effects that can happen during the development.

Finally, the aim of the team, is to have a smart steering system on the roads, save lives with the development of safety critical systems, which cannot be fulfilled without finding the bugs in the system first.

In case of testing the Budapest group focuses on the verification and validation test activities. Which involves the process of the followings:




However, considering testing as a big group, there are three kinds of testing:



My capstone project focuses on the component testing lifecycle, on the work of the Software System Test Group. The focus: verification of functionalities and interfaces of the software components. Tests are executed on real target processor environment.

The Software System Test Group itself is divided into two teams:

- Software Safety Test Team
- Software Integration Test Team

A few words about the team testing processes:

- The teams have a matrix structure, for each testing project they nominate a Test Project Leader.
- The first round of software testing is unit testing. The focus is on specific units or components of the software to determine whether each one is fully functional. The main aim of this endeavour is to determine whether the application functions as designed. In this phase, a unit can refer to a function, individual program or even a procedure. The unit testing is done by either the white or black box unit testing. In case of black box approach, the tester is unconcerned about the internal structure and workings of the software. Test data is derived solely from the specification. Whereas in case of the white box approach, the tester analysis the internal logic of the software under test. White- box module tests are directed toward code coverage. Following this, the next step is the software system test; the hardware environment, the hardware in the loop simulator stage. The unit testing is followed by integration testing to find interface defects between the modules/functions. Next, system testing, where the complete application is tested as a whole. Lastly, the acceptance testing, in this stage they are determining whether the system is ready for release.

From this short summary it can be concluded that the software testing is a complicated and a far- reaching process. One of the biggest challenges of testing is the documentation and recording of the processes. Unit case testing for example requires multiple developers to work at the same time, with different approaches. This leads us to ask *How can we track, and record different testing process in a collaborative software development team?* Version management systems, seem to offer a solution to this problem.

## SVN Tortoise Version Management with Subversion

For version management, Thyssenkrupp uses Subversion. In the collaborative software development area, it is essential to have a version control system. It is a server application,

which stores folders and files like a filesystem. Apache Subversion for is example (or SVN, derived from the command) is a widely used version control system the manages the various versions of source codes, web pages, and documentations. It stores changes made to the files and directories, thus making it possible to recover the project from a previous state. With SVN, multiple users can work on the same project, using project repositories and access previous states (older versions) whenever it is needed.

The repositories are storage databases. The files are stored on the server and can be downloaded if necessary - the version on the server is not modified, only the users can modify it in the local content, which are called working copies of their server versions. If the user is done with the changes, they can upload their work to the server. The upload of the working copy called “commit “. The repository has a special layout / project structure disregarding the involved process, there are some standard, recommended ways to organize a repository. For every project a trunk directory can be created to hold the “main line” of development, a branches directory to contain branch copies, and a tags directory to contain tag copies. Branches are the forks of the trunk, it is usually used for testing new features, until they are merged back to the trunk. If a developer has more time-consuming work, he is able to create a branch, and work on it individually, without disturbing other developers. When he finished with his own version, he can merge the changes to the trunk. Further, branching can be used when the developer wants to create a software variant, with side changes, but in that case, they are not merging the branch back to the trunk. The tags are snapshots of the trunk, they are SVN’s baseline. They are used, to label specific revisions, and store them for later access, the method is to create these directories for the smallest unit of production, called a project.

TKP uses the subversion project structure (trunk, branch, tag) in different processes, not only in software testing. It is used in the field of product release, software construction, software integration testing, and in configuration management.

For my capstone project, it was crucial to gain in dept knowledge regarding SVN, as I have worked with test logs located in SVN repositories.



## The project

During the development of the automatized software testing there are two jobs that can be done: develop a test from zero, or adapt (or in other words, “port”) an existing version. Throughout the years, the team had recorded how much time they spent, and how much work was done at all together, but they forgone measurement, the number of testing methods, and the cases when each test was used on the various different methodologies.

The design and the project's key objective were to be able to explain based on the test logs, which of the tests, had been developed from 0, and which of them was done with porting. After sorting the data table by date, a special test id occurred on some of the projects. In the instances where the identification number occurred, those tests were developed from zero, and the others were done by porting. The test logs and projects were in SVN, in the format of zip files. Those zip files contained hundreds of xml files, with thousands of rows. My task was to unzip all the files of the data archive, process the content, and put the required information from them into corresponding data tables. The processed file projects are the results of the test logs, which have captured different testing's. During the test runs, the automatized scripts are running software testing, for electrical power assisted steering systems. It is a requirement based testing, the scripts were developed, for functional testing requirements. The files are involving enormous amount of data such as the testing type, date, test creator ID, and how many hours were spent with the testing. However, the most crucial points of the test logs are the following:

- Test ID, based on the ID we can look up in other documentation, that what kind of software requirement or functionality was tested.
- If the test is passed or fail.
- Also, captures additional information for the evaluation of the tests: such as expected behaviour of the software (e.g.: value of the output, reaction time), moreover the actually observed behaviour during the test runs, and there is some plus data which helps the tester to understand what has happened under the running, called hardware signs time rows.

Subsequently the running, the software testing team checks the test logs. In case of they have a failed test, they raise an error ticket. The error ticket includes the error and the test log as well. Afterwards, all of the test logs are archived, which gave me the opportunity to proceed them.

## Conclusion

In conclusion, my capstone project was a challenging but useful task. Firstly, the engineering company with the focus of electrical power assisted steering systems and software testing, was a completely new field for me. The first milestone was to understand the Tortoise repository system, also the file system commands, to be able to build a script which can access to it. Not to mention the data access difficulties. The data archive which I worked on was only accessible through company device with company network connection. Moreover, the project itself was deeply thought-provoking, as it required a higher level of information technology knowledge. Still, after all of the obstacles, I feel I gained a lot with my capstone project. I observed new areas of business, and I made some good connection inside the company. Finally, with my project I deepened my programming knowledge in Python more than ever before. On the following pages, I have illustrated my script for the project, expended with some comments.

```
import re
import os
import sys
import zipfile
import subprocess
from shutil import rmtree
from ast import literal_eval
from datetime import datetime
from subprocess import Popen, PIPE
from xml.etree.ElementTree import parse, Element


class TestParser:

    def __init__(self, SVN_folder = 'C:\\Program Files\\TortoiseSVN\\bin\\svn.exe', \
                 log_path = 'http://d1dapsvn01/svn/SwSystemTestR8R9/50_CustomerProjects/', \
                 destination = 'C:/SVNOut', \
                 pattern = ".*/tags/.*/TLOG_[^/]*_SWIT.zip$", \
                 meta_data = ['projectID', 'subProjectID', 'platformSw', 'swRelease']):
        self.SVN = SVN_folder
        self.logs = log_path
        self.dest = destination
        self.pattern = pattern
        self.meta = meta_data

        #create destination folder, if it does not exist
        if not os.path.exists(self.dest):
            os.mkdir(self.dest)

    def main_process(self):

# ============================================================
# extract an existing zip file and list some file names      #
# ============================================================

# define file name pattern for swit log files
self.myfile = re.compile( self.pattern )

# main loop
self.subp = Popen( [self.SVN, 'list', '-R', self.logs], stdout=PIPE, shell=True )

for tline in self.subp.stdout:
    tline = tline.rstrip()        # chop off trailing newline
    tline = tline.decode('utf-8')
    #print(tline)
    if self.myfile.match( tline ):
        # process here the zip file
        self.zippath = self.logs + '/' + tline
        print self.zippath
        self.localzip = self.dest + '/' + self.zippath.split("/")[-1]
        #Export matching files to a local folder
        self.subpexp = Popen([self.SVN, "export", self.zippath, self.dest], stdout = PIPE, shell = True)
        #blocks timeout error and lets the process run
        self.subpexp.wait()
        #calling function which unzip files to a subfolder
        self.read_logs(self.localzip, self.dest, self.zippath)
        #delete the zip file from the local folder
        os.remove(self.localzip)

print "Process successfully ended at " + str(datetime.now())
```

```
def read_logs(self, file, folder, svnfile):
    # The file we parsing
    self.fname = file
    self.svnfile = svnfile
    # Set the directory you want to start from
    self.rootfolder = folder
    self.dname = self.rootfolder + "/extr"

    # delete working directory, if it exists to remove previously extracted files
    if os.path.exists(self.dname):
        rmtree(self.dname)

    #create the working directory
    os.mkdir(self.dname)

    #extract the files from the zip
    self.extract(self.fname, self.dname)

    #Looping through the extracted file structure
    for dirName, subdirList, fileList in os.walk(self.dname):
        for fname in fileList:
            if fname[-4:] == ".xml":
                #if the file is xml, create the path to it
                self.xmlfile = os.path.join(dirName,fname)
                #Parsing the xml file
                self.xml_result = self.xml_parser(self.xmlfile)
                #If there is element in the xml_result list, write it to the file
                if self.xml_result:
                    #open and write text file if tc_list has content.
                    self.writefile(self.rootfolder, self.xml_result)

    #delete the working directory to remove previously extracted files
    rmtree( self.dname )

def extract(self, file, folder):
    #Extract the zip file to the given local folder
    self.zip_ref = zipfile.ZipFile(file, 'r')
    self.zip_ref.extractall(folder)
    self.zip_ref.close()

def xml_parser(self, xmlfile):
        #Parsing the relevant xml file
        self.xmlf = xmlfile
```

```
#Result list, where we keep the relevant records;
#dictionary from which we get the meta data,
#result strings to save into the testresult.txt
self.tc_list = []
self.conf_dict = {}
self.conf_str = ""
self.tc_ecu = "Null"



#Simple exception handling: if xml is unstructured,
#then problem occures right when calling parse method
#If there is an error, we print out file name to the terminal,
#and continue the process --> create error log
try:
    #Calls Parse method from ElementTree
    self.x = parse(self.xmlf)
except:
    print "Unstructured file: " + self.xmlf
    #This return terminates the function by returning an empty list
    self.write_error(self.rootfolder, self.svnfile)
    return self.tc_list

#Find and format start time
for item in self.x.iterfind('start_time'):
    #get the time string
    self.raw_date = item.text.split(',')[0]
    #Calling time format function which contains the exception handling,
    #for that one particular timestamp
    self.tc_date = self.time_format(self.raw_date)

#Handling <config> text as dictionary
#config is always present, no need to deal with config2
for item in self.x.iterfind('config'):
    self.conf_dict = literal_eval(item.text)
    for key in (self.meta):
        if key in self.conf_dict:
            self.conf_str = self.conf_str + self.conf_dict[key] + '\t'
        else:
            self.conf_str = self.conf_str + 'Null' + '\t'

for item in self.x.iterfind('ecu_sw_ver'):
    self.tc_ecu = item.text.split(',')[0]
    if "TKP*INFO" in self.tc_ecu:
        self.tc_ecu = self.tc_ecu.split(':')[1] + '\t'
    else:
        self.tc_ecu = self.tc_ecu + '\t'

self.united_config = self.conf_str + self.tc_ecu
```

```
#item is the relevant parent tag.
for item in self.x.iterfind('testsuite'):
    self.tc_trs = item.findtext('tc_trs')
    self.tc_id = item.findtext('tc/tc_id')
    #tc_id text value of the relevant tag
    self.tc_verdict = item.findtext('tc/tc_verdict')
    #tc_verdict text value of the relevant tag
    #If tc_verdict pass or fail, append it to the result list
    if self.tc_verdict in ["pass", "fail"]:
    self.tc_list.append('\t'.join(e for e in
    [self.tc_trs, self.tc_verdict,  self.tc_date, self.united_config])
    + self.tc_id + '\n')


#Function returns the result list, regardles it has content or not
return self.tc_list

def time_format(self, date):
    self.r_date = date
    #There is only one renegade time stamp, which we have to handle
    try:
        #convert to datetime
        self.to_date = datetime.strptime(self.r_date, '%Y %b %d %H:%M:%S')
        #convert back to string in desired format
        self.tc_date = self.to_date.strftime('%Y-%m-%dT%H:%M:%S')
    except:
        #convert to datetime
        self.to_date = datetime.strptime(self.r_date, '%a %b %d %H:%M:%S %Y')
        #convert back to string in desired format
        self.tc_date = self.to_date.strftime('%Y-%m-%dT%H:%M:%S')
        pass

    return self.tc_date

def writefile(self, rootfolder, list_to_write):

    #the Rootfolder contains the test result file
    self.textpath = rootfolder + "/testresult.txt"
    #if the file doesnt exists, create it,
    #and write the result list's content into it.
    self.lst = list_to_write
    if not os.path.exists(self.textpath):
        with open(self.textpath, 'w+') as f:
            for a in self.lst:
                f.write(a)
    #if exists, then just write the result list's content into it.
    else:
        with open(self.textpath, 'a+') as ff:
            for a in self.lst:
                ff.write(a)
```

```
def write_error(self, rootfolder, unstructured):

    #the Rootfolder contains the test result file
    self.textpath = rootfolder + "/errorlog.txt"
    #if the file doesnt exists, create it,
    and write the result list's content into it.
    if not os.path.exists(self.textpath):
        with open(self.textpath, 'w+') as f:
            f.write(unstructured + '\n')
    #if exists, then just write the result list's content into it.
    else:
        with open(self.textpath, 'a+') as ff:
            ff.write(unstructured + '\n')

if __name__ == "__main__":

    TestParser().main_process()
```



CEU Department of Economics and Business
CENTRAL EUROPEAN UNIVERSITY