Hacettepe University

Department of Computer Engineering

Doctor’s Aid Assignment Report

Course Name:

BBM103 – Introduction to Programming Lab

Assignment – 2

Mert ERGÜN - B2220356062

24.11.2022

Table of Contents

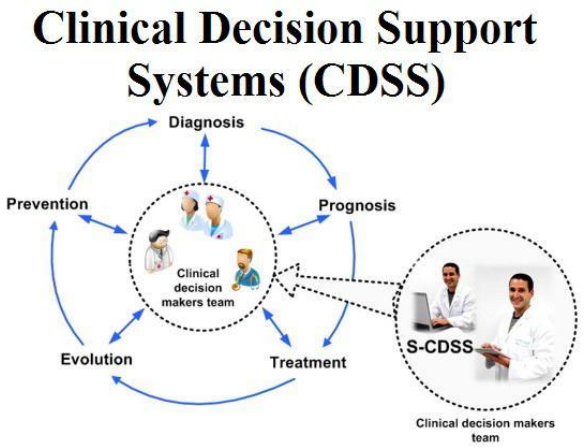
1- Introduction

In this assignment we are asked to design a simple kind of Clinical Decision Support System (CDDS).

CDSS is an application designed to provide clinicians, staff and patients with specific health information. CDSS encompasses a variety of tools to enhance decision-making in the clinical workflow. These tools include automatic alerts and reminders to care providers and patients, clinical guidelines, condition-specific order sets; focused patient data reports and summaries, documentation templates, diagnostic support, and contextually relevant reference information, among other tools.

Basically, CDSS is a very good example of how computer artificial intelligence can be used optimally in healthcare. The success of the artificial intelligence theme in improving human life conditions can come to the forefront thanks to CDSS.

The Clinical Decision Support System that we were asked for in this assignment is a probabilistic system called "Doctor's Aid" that focuses directly on cancer diseases.



2- Analysis

The problem to be solved during this assignment is to reduce the burden on clinicians and doctors in diagnosing and treating cancer in individuals with cancer and to make diagnoses and treatments more effective and successful.

When diagnosing cancer, the accuracy of the diagnosis is key. Since starting the wrong treatment will negatively affect the patient's future, the main goal during this assignment is to prepare a Clinical Decision Support System that can work properly enough to prevent misdiagnosis and treatments, and to examine the patient data entered and make recommendations to clinicians about treatment.

The patient data to be entered into the program by the user (clinician) are; "Patient Name", "Diagnosis Accuracy", "Disease Name", "Disease Incidence", "Treatment Name" and "Treatment Risk".

The entered data is saved in the database. Through the data saved specific to the patient's name, information can be presented to the user, from the possibility of this patient actually having the disease in question to whether the treatment should be performed or not.

The inputs that can be entered into the program are as follows:

- Create
- Remove
- Probability
- Recommendation
- List

The outputs that the program can output are as follows:

- Patient … recorded
- List of patients.
- Patient … has a probability of … of having …
- System suggest … to have the treatment.
- System suggest … NOT to have the treatment.

3- Design

The code I wrote for the designed program consists of 8 different functions, each of which works in conjunction with each other. Even though each function's task is independent of each other, they connect to each other to work. It analyzes the input data and writes on the output file according to the data and the clinician's requests.

3.0 – Finding Read and Write Files

The input and output files must be in .txt format in the file location where the program is located. All inputs to the program will be in the input file and all outputs from the program will be in the output file. The program does not actively print output to the screen and does not receive input from the screen. Accordingly, the location and names of the input and output files must be defined in the program.

The library named "os" is used for this. The location where the file is running on the operating system is found and the input and output files are searched in this location.

```
# Importing OS module for getting current directories for read and write files.
import os

current_dir = os.getcwd()  # We will use this to get the current directory for read and write files
read_file_name = "doctors_aid_inputs.txt"  # The name of the file to read
write_file_name = "doctors_aid_outputs.txt"  # The name of the file to write
read_file_path = os.path.join(current_dir, read_file_name)  # The path to the file to read
write_file_path = os.path.join(current_dir, write_file_name)  # The path to the file to write
```

3.1 – Reading File

First of all, the designed program needs an "Input.txt" file to receive the input data. The program should open this file, examine each line in it individually to get the commands entered, pass the commands to the necessary functions and print the data returned from the functions to another file.

A function called "read_file()" was written to solve this problem. The function is looking for a "doctors_aid_inputs.txt" file in the location where the program exists. It opens the input file and examines the commands line by line starting from the first line.

The way the input file is written;

"command argument"

```
        def read_file():
    with open(read_file_path, 'r') as f:
        for lines in f:
            if lines.startswith("create "):
                create_pat(lines)
            if lines.startswith("remove "):
                remove_pat(lines)
            if lines.startswith("probability "):
                check_probability(lines)
            if lines.startswith("recommendation "):
                recommendation(lines)
            if lines.startswith("list"):
                getlist()
```

The function examines the line, looking at the first word, "Command". According to the command, it calls the next required function.

3.2 – Writing File

After the designed program performs the necessary operations in response to the entered data, it should print the outputs into a file named "doctors_aid_outputs.txt" in the same location as the program.

For each input, at least one output is required. Each function that performs an operation calls the write function (write_to_file()) at the end.

```
# Our write to file function that writes "text" variable to the
"write_file".
def write_to_file(text):
    with open(write_file_path, "a") as s:
        s.write(text)
```

3.3 – Creating Patients

Each command from the input file must execute a function on the program. A new patient needs to be created as a basic command. The new patient to be created will be both saved in the database and certain values related to the patient will be calculated.

Create command is used to add a new patient to the system. The usage of the command is to enter the name of the patient to be added after the command and then

enter the required data. This input creates a patient with the name given after the command.

After creating the patient and adding it to the database, it prints an output like "Patient {} is recorded." to the output file.

Patients' data is stored in multidimensional lists. This multidimensional list, referred to as pat_list in my code, consists of a separate list for each patient. These lists hold the data entered by clinicians when creating a patient.

This function also calls another (probability calculation) function. It reduces the computational burden by calculating the probability of disease when creating a patient. It also performs the probability calculation for each new patient created.

3.4 – Removing Patients

The "remove" command is used when the patients added to the database need to be deleted afterwards. After this command, the patient matching the given name is deleted from the program database. All information about this patient is first deleted from the "Patient List", which is the main database, and then other calculated data about this patient (disease probability, etc.) are also deleted from their own data systems, so that nothing is left about the patient.

The remove() function is activated immediately after seeing the remove command in the input file, it takes the name after the remove command and searches the list named pat_list to see if this patient exists. If the patient exists, it deletes all data related to the patient. If the patient does not exist, the function terminates with a warning that the patient does not exist.

3.5 – Patient’s Probability of Having Disease

Using the "probability" command, the probability that this patient has the cancer in question can be calculated based on the name of the patient that follows. This calculation is based on several things. Basically, using the confusion matrix, this

calculation is based on the probability of actually having the disease, apart from the accuracy of the diagnosis.

This calculation is based on the data entered when the patient was first registered. Based on this data, the diagnostic accuracy and disease incidence are taken as a basis, and after a few manipulations, the actual disease probability is calculated. The data obtained from this calculation can then be used to recommend treatment.

The function first checks whether the name entered after the probability command is in the pat_list. If the name is in the patient list, it proceeds to the second step and finds false positives based on the accuracy of the diagnosis and true positives based on the incidence of the disease. After a few mathematical operations, it finds the true disease probability and stores it in a "dictionary" called "disease probability". The data stored in this dictionary, along with the patient's name, can then be easily retrieved.

The probability function is executed not only when called in the input file, but also when a new patient is created. In this way, the probability of disease is calculated for each patient without the need to call the probability function manually.

Two different probability functions were used to integrate this into the code. The first function is called when the patient is created and only calculates the probability of disease. The second function is called when the probability command is entered and prints the disease probability data for that patient to the output file.

```
# This is our "Probability" function that checks for probability of disease
in that patient.
def check_probability(x):
    pat = x.split()[1]
    counter_p = 0
    # Checks for patients name in database and if finds it, then writes its
probability to file.
    for i in pat_list:
        if pat in i:
            write_to_file("Patient {} has a probability of {}% of having
{}.\n".format(i[0], dp[i[0]], i[2]))
        else:
            counter_p += 1
        if counter_p == len(pat_list):
            write_to_file("Probability for {} cannot be calculated due to
absence.\n".format(pat))
            counter_p = 0
```

```
# Check Probability Function uses Confusion Matrix to calculate probability
of disease in that patient when it has been added to the patient list.
def probability(line):
    tp = float(line[4])
    dp[line[1]] = round(tp / (tp + fp[line[1]]) * 100, 2)
    if float(dp[line[1]]) == int(dp[line[1]]):
        dp[line[1]] = int(dp[line[1]])
```

3.6 – Recommendation for Particular Treatment

The function "recommendation", which performs the program's main task of providing recommendations. It allows the program to suggest to the user whether the name that follows the recommendation command in the input file should receive the treatment in question.

Clearly, the accuracy of this proposal is vitally important. The program needs to work correctly and control for the risk of treatment and the probability of disease. As a result of the audit of treatment risk and disease probability, it gives feedback to the clinician about whether it is right or wrong to perform this treatment on the patient in question.

After the "recommendation" command, the function retrieves the actual disease probability of this patient by looking up the name entered in the "disease probability" dictionary. Then it finds the treatment the patient will receive and the risk of this treatment in the list named pat_list. It compares the disease probability and the treatment risk.

If the treatment risk is higher than the disease probability, the treatment should NOT be applied.

If the risk of the disease is higher than the risk of the treatment, the treatment should be applied.

```
# This is our "Recommendation" function that checks for recommendation for
we should treat the patient or not.
def recommendation(x):
    line = x.split()
    pat = line[1]
    counter_rc = 0
    # Checks that if "Disease Probability" is higher than the "Treatment
Risk". If it is, suggests us to treat.
    for i in pat_list:
        if pat in i:
            if dp[i[0]] > risk[i[0]]:
```

```
                write_to_file("System suggests {} to have the
treatment.\n".format(pat))
            else:
                write to file("System suggests {} NOT to have the
treatment.\n".format(pat))
        else:
            counter_rc += 1
        if counter_rc == len(pat_list):
            write_to_file("Recommendation for {} cannot be calculated due
to absence.\n".format(pat))
```

3.7 – Listing Patients

Provides an overview and listing of all patients in the database with their information. Prints the names of the patients and the data entered when creating the patient as a list on the screen. It has no function other than printing a list to the output file. It does not accept any arguments. It only works on typing "list" and lists all patients currently in the database.

Different alignment techniques are applied according to the data entered to align the list. Longer names have more spaces and shorter names have shorter spaces.

The function name "list" is defined as get_list() to avoid confusion with the built-in list function in Python.

```
def getlist():

write_to_file("Patient\tDiagnosis\tDisease\t\t\tDisease\t\tTreatment\t\tTre
atment\nName\tAccuracy\tName\t\t\t"
                  "Incidence\tName\t\t\tRisk\n----------------------------
-----------------------------------------\n")
    for pat in pat_list:
        if len(pat[4]) == 16:
            write_to_file("{}\t{}\t\t{}\t{}\t{}{}\n".format(pat[0], pat[1],
pat[2], incidence[pat[0]], pat[4], pat[5]))
        elif len(pat[4]) == 7:
            write_to_file("{}\t{}\t\t{}\t{}\t{}\t\t\t{}\n".format(pat[0],
pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))
        elif len(pat[2]) == 11:
            write_to_file("{}\t{}\t\t{}\t\t{}\t{}\t{}\n".format(pat[0],
pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))
        elif len(pat[0]) == 2:
            write_to_file("{}\t\t{}\t\t{}\t{}\t{}\t{}\n".format(pat[0],
pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))
        else:
            write_to_file("{}\t{}\t\t{}\t{}\t{}\t{}\n".format(pat[0],
pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))
```

4 – Programmer's Catalogue

Python 3.10 code for "Doctor's Aid" program is:

```
# Importing OS module for getting current directories for read and write files.
import os

current_dir = os.getcwd()  # We will use this to get the current directory for read and write files
read_file_name = "doctors_aid_inputs.txt"  # The name of the file to read
write_file_name = "doctors_aid_outputs.txt"  # The name of the file to write
read_file_path = os.path.join(current_dir, read_file_name)  # The path to the file to read
write_file_path = os.path.join(current_dir, write_file_name)  # The path to the file to write
pat_list = []  # List of patients, most important list because it holds all the data of the patients
fp = dict()  # Dictionary for "False positive" of patients
dp = dict()  # Dictionary for "Disease probability" of patients
risk = dict()  # Dictionary for risk of treatment
incidence = dict()  # Dictionary for incidence.


# Our write to file function that writes "text" variable to the "write_file".
def write_to_file(text):
    with open(write_file_path, "a") as s:
        s.write(text)


# This is our "Create Patient" function that creates patients and computes their "Disease Probabilities".
def create_pat(x):
    line = x.split()  # First it splits the line into words.
    for i in range(len(line)):  # Gets rid of the ","s.
        line[i] = line[i].strip(',')
    for pat in pat_list:
        if line[1] in pat:
            write_to_file("Patient {} cannot be recorded due to duplication.\n".format(line[1]))
            return
    if line[6] == 'Targeted':  # This is for 'Bugfix', if it sees that there is a word "Targeted", it compiles it with "Therapy"
and makes them one word.
```

```
        line[6:8] = [' '.join(line[6:8])]
    line[3:5] = [' '.join(line[3:5])]  # This line combines the two words (Cancer name and cancer).
    fp[line[1]] = round(1 - float(line[2]), 4)  # Computes False Positive and rounds it to 4 decimals.
    risk[line[1]] = float(line[6])*100  # Computes risk.
    line[6] = "{:.0%}".format(float(line[6]))  # Turns risk to percentage.
    line[2] = "{:.2%}".format(float(line[2]))  # Turns Diagnosis Accuracy to percentage.
    incidence[line[1]] = line[4]  # Saves incidence for later uses.
    line[4] = line[4].split('/')  # Turns incidence to float number.
    divisor = float(line[4][0])
    dividend = float(line[4][1])
    line[4] = divisor/dividend
    pat_list.append(line[1:])  # Adds patient to the patient list.
    write_to_file("Patient {} is recorded.\n".format(line[1]))  # Writes message to file.
    probability(line)  # Checks probability.


# This is our "Remove Patient" function that removes patients from pat_list.
def remove_pat(x):
    line = x.split()
    counter_r = 0  # Counter for 'Error Messages'.
    for i in range(len(line)):
        line[i] = line[i].strip(',')
    # Searches for patients and if catches, deletes it from all database.
    for pat in pat_list:
        if line[1] in pat:
            write_to_file("Patient {} is removed.\n".format(line[1]))
            pat_list.remove(pat)
            del dp[pat[0]]
            del fp[pat[0]]
            del risk[pat[0]]
            del incidence[pat[0]]
        else:
            counter_r += 1
        if counter_r == len(pat_list):  # Checks if counter is equal to len(pat_list) and if it is equal, then it means that there is no patient in pat_list with that name.
            write_to_file("Patient {} is cannot be removed due to absence. \n".format(line[1]))
            counter_r = 0
```

```
# This is our "Probability" function that checks for probability of disease in that patient.
def check_probability(x):
    pat = x.split()[1]
    counter_p = 0
    # Checks for patients name in database and if finds it, then writes its probability to file.
    for i in pat_list:
        if pat in i:
            write_to_file("Patient {} has a probability of {}% of having {}.\n".format(i[0], dp[i[0]], i[2]))
        else:
            counter_p += 1
        if counter_p == len(pat_list):
            write_to_file("Probability for {} cannot be calculated due to absence.\n".format(pat))
            counter_p = 0


# Check Probability Function uses Confusion Matrix to calculate probability of disease in that patient when it has been added to the patient list.
def probability(line):
    tp = float(line[4])
    dp[line[1]] = round(tp / (tp + fp[line[1]]) * 100, 2)
    if float(dp[line[1]]) == int(dp[line[1]]):
        dp[line[1]] = int(dp[line[1]])


# This is our "Recommendation" function that checks for recommendation for we should treat the patient or not.
def recommendation(x):
    line = x.split()
    pat = line[1]
    counter_rc = 0
    # Checks that if "Disease Probability" is higher than the "Treatment Risk". If it is, suggests us to treat.
    for i in pat_list:
        if pat in i:
            if dp[i[0]] > risk[i[0]]:
                write_to_file("System suggests {} to have the treatment.\n".format(pat))
            else:
                write_to_file("System suggests {} NOT to have the treatment.\n".format(pat))
        else:
            counter_rc += 1
        if counter_rc == len(pat_list):
```

```
            write_to_file("Recommendation for {} cannot be calculated due to absence.\n".format(pat))


# This function just gets the list of patients. It has exceptions for longer or shorter words.
def getlist():
    write_to_file("Patient\tDiagnosis\tDisease\t\t\tDisease\t\tTreatment\t\tTreatment\nName\tAccuracy\tName\t\t\t"
                  "Incidence\tName\t\t\tRisk\n-------------------------------------------------------------------------------\n")
    for pat in pat_list:
        if len(pat[4]) == 16:
            write_to_file("{}\t{}\t\t{}\t{}\t{}{}\n".format(pat[0], pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))
        elif len(pat[4]) == 7:
            write_to_file("{}\t{}\t\t{}\t{}\t{}\t\t\t{}\n".format(pat[0], pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))
        elif len(pat[2]) == 11:
            write_to_file("{}\t{}\t\t{}\t\t{}\t{}\t{}\n".format(pat[0], pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))
        elif len(pat[0]) == 2:
            write_to_file("{}\t\t{}\t\t{}\t{}\t{}\t{}\n".format(pat[0], pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))
        else:
            write_to_file("{}\t{}\t\t{}\t{}\t{}\t{}\n".format(pat[0], pat[1], pat[2], incidence[pat[0]], pat[4], pat[5]))

def read_file():
    with open(read_file_path, 'r') as f:
        for lines in f:
            if lines.startswith("create "):
                create_pat(lines)
            if lines.startswith("remove "):
                remove_pat(lines)
            if lines.startswith("probability "):
                check_probability(lines)
            if lines.startswith("recommendation "):
                recommendation(lines)
            if lines.startswith("list"):
                getlist()


# This part is our main function, which is responsible for reading the file and leading us to the wanted functions.
if __name__ == "__main__":
    read_file()
```

 # Mert ERGÜN

Time Spent for Analysis	The analysis part actually consisted of understanding the problem, doing research on CDSS, understanding the Confusion Matrix theme and analyzing the input file. The most time-consuming parts of this part were examining the types of inputs that could be entered, understanding the sample outputs that the program should give and implementing them in the file. Examining wikipedia data on Confusion Matrix, watching videos on the subject and reading 2 short articles helped in understanding the theme. By the time the analysis process was completed, I had a full knowledge of the program to be prepared. This whole process took about 3-4 hours, with intervals.
Time Spent for Design	The design part was obviously the most difficult part of this assignment. Not only did it take a long time, but the effort to solve the dozens of problems encountered made this process very valuable. As a result of the research I did to solve the problems I encountered, I learned dozens of things I never knew. In addition to contributing to my algorithm-based understanding, it also improved my problem solving skills. In this process, I tried to understand the problem, implement what I was asked to implement in the program and solve the problems that arose while doing these. When all these stages are examined, I can say that the design process took about 8 hours with intervals.
Time Spent for Reporting	Above all, the process that scared me the most during this assignment was the reporting part. Before I started reporting, I tried to write the code completely and clean up any errors that might occur. I thought that the reporting process would be very difficult and long, so I waited for the right time to start. Finally, after making sure that my code was solid, I started the reporting process. Another point that improved me in this assignment was the reporting process. In order to design this report, I analyzed other reports, read pages and pages of sample reports and started writing my report with a dictionary. In order to use the right terms and avoid making mistakes, I could only write short sentences in a very long time. Considering this process, it took about 10 hours to write the report.

 # b2220356062

4.1 – Reusability of Program

The designed program basically receives randomly typed inputs from the input file, processes them with the above-mentioned functions, saves some data in the database and prints the output to the output file. Since each function is defined separately from each other, they can easily be taken and implemented into another program. The variables used in the functions are named as clearly as possible so that it is clear which variable is used for what purpose. The comment lines embedded in the code are designed to explain the necessary points. Nevertheless, since each function is generally interdependent, if one is missing, the others will not work properly.

When the program is run, it starts automatically to open the input file directly and to write the output to the output file. For this reason, the defined function "read_file()" is already called in "main". This function calls other functions according to the input file. The user does not need to call an extra function.

5 – User's Catalogue

5.1 – User Manual

The clinician who will use the program must first make sure that there are two files named "doctors_aid_inputs.txt" and "doctors_aid_outputs.txt" in the location where the program is running.

In the "doctors_aid_inputs.txt" file, all inputs to be entered into the program are written (as described below).

The file "doctors_aid_outputs.txt" is left empty. After the program processes all the data it will receive from the input file, it will present the outputs to the user through this file.

The commands that the user can enter into the program and their functions are as follows:

create -> Allows to create a patient.

remove -> Removes the entered patient from the database.

list -> Lists the existing patients in the database.

probability -> Shows the disease probability of the entered patient.

recommendation -> Shows the recommendation on whether the entered patient should be treated or not.

5.1.1 – Create Command

The create command allows a new patient to be saved in the database. The correct usage of the command is as follows:

create [Patient Name], [Diagnosis Accuracy (in form of decimal)], [Disease Name], [Disease Incidence (in form of integer division)], [Treatment Name], [Treatment Risk (in form of decimal)]

Example:

create Hayriye, 0.999, Breast Cancer, 50/100000, Surgery, 0.40

The output will be “Patient Hayriye is recorded.” if you wrote the command right.

The output will be “Patient Hayriye cannot be recorded due to duplication” if you try record more than one Hayriye.

Misuses of the create command;

Trying to register another patient with the same name as an already existing patient. -> The program warns the user in this case and does not add the new patient, saying that the patient already exists.

Entering more or less than one or more of the required data. -> The program closes with an error in this case. Do not enter missing data!

5.1.2 – Remove Command

The remove command deletes a patient in the database. The correct usage of the command is as follows:

remove [Patient Name]

Example:

remove Ateş

The output will be “Patient Ateş is removed” if you wrote command right.

The output will be “Patient Ateş cannot be removed due to absence.” if you try to remove Ateş while there is no patient named Ateş.

5.1.3 – List Command

The list command lists all patients in the database. The correct usage of the command is as follows:

list

Example:

list

The output will be:

```
Patient Diagnosis   Disease         Disease     Treatment         Treatment
Name    Accuracy    Name            Incidence   Name              Risk
-------------------------------------------------------------------------
Hayriye 99.90%      Breast Cancer   50/100000   Surgery           40%
Ahmet   98.60%      Prostate Cancer 21/100000   Targeted Therapy52%
Deniz   99.99%      Lung Cancer     40/100000   Radiotherapy      50%
Ateş    99.00%      Thyroid Cancer  16/100000   Chemotherapy      2%
```

5.1.4 – Probability Command

The probability command calculates the actual probability of disease for the desired patient. The correct usage of the command is as follows:

probability [Patient Name]

Example:

probability Deniz

The output will be “Patient Deniz has a probability of %80 of having Breast Cancer.” if you wrote it right. In order for this command to work, the patient whose probability is desired must be in the database, that is, it must have already been added using the create function.

The output will be “Probability for Deniz cannot be calculated due to absence.” if there is no such a patient in database.

5.1.5 – Recommendation Command

The “recommendation” command will be used to give a system recommendation to a patient about whether have the treatment. The correct usage of the command is as follows:

recommendation [Patient Name]

Example:

recommendation Hayriye

The output will be “System suggest Hayriye NOT to have the treatment.” if Hayriye’s Disease Probability is lower than suggested treatment’s risk.

The output will be “System suggest Hayriye to have the treatment.” if Hayriye’s Disease probability is higher than suggested treatment’s risk.

The output will be “Recommendation for Hayriye cannot be calculated due to absence.” if there is no such a patient named Hayriye.

6 – Grading Table

Evaluation	Points	Evaluate Yourself / Guess Grading
Indented and Readable Codes	5	5
Using Meaningful Naming	5	5
Using Explanatory Comments	5	5
Efficiency (avoiding unnecessary actions)	5	3
Function Usage	25	25
Correctness	35	32
Report	20	20
There are several negative evaluations	…	…