School of Engineering and Information Technology

ASSESSMENT COVER SHEET

Student Name	Ruchan Suwal
Student ID	S278220
Assessment Title	Assignment 1
Unit Number and Title	PRT 452 / Software Engineering: Process and Tools
Lecturer/Tutor	Mr. Kai Wang
Date Submitted	5/09/2016
Date Received	

KEEP A COPY

Please be sure to make a copy of your work. If you have submitted assessment work electronically make sure you have a backup copy.

PLAGIARISM

Plagiarism is the presentation of the work of another without acknowledgement. Students may use a limited amount of information and ideas expressed by others but this use must be identified by appropriate referencing.

CONSEQUENCES OF PLAGIARISM

Plagiarism is misconduct as defined under the Student Conduct By-Laws. The penalties associated with plagiarism are designed to impose sanctions on offenders that reflect the seriousness of the University's commitment to academic integrity.

I declare that all material in this assessment is my own work except where there is a clear acknowledgement and reference to the work of others. I have read the University's Academic and Scientific Misconduct Policy and understand its implications.*

http://www.cdu.edu.au/governance/documents/AcademicandScientificMisconductPolicyv1.03Jan2011.pdf.

Signed… RUCHAN SUWAL……..Date……5/09/2016

*** By submitting this assignment and cover sheet electronically, in whatever form you are deemed to have made the declaration set out above.**

Q1. Create a SVN or Git directory for you assignment (including word or pdf documents and programming code);

https://github.com/suwalruchan/Software-Engineering-Assignement-1

Q2. Some English words can be transferred to another one by reorder the letters.

For example, 'how' and 'who'.

Requirements:

- **Input**: A number of words, each word will be in one line;
- **Output**: Print out the number of words in each group and the words in the group ordered by the number of words in the group from the largest to the smallest.

For example: how

who
here
paw
wap
awp

- **Output:**

3: paw, wap, awp
2: how, who
1: here

1) Write a program to group the similar words and print out the number of words in each group and the words in the group ordered by the number of words in the group from the largest to the smallest;

```
class Anagram
	attr_accessor :inputs, :outputs

	def initialize
		@outputs = []
		puts "Enter words and enter blank to view result"
		@inputs = []
		input = gets.chomp
		terminate_loop = false
		@inputs << input
		while(!terminate_loop) do
			input = gets.chomp
			terminate_loop = input.empty?
			@inputs << input
		end
		@inputs = @inputs.reject(&:empty?)
	end
```

```
	def split_words(input)
		input.split('')
	end

	def compare
		inputs.each do |input|
			splitted_input = split_words(input)
			unless outputs.empty?
				outputs.each do |output|
					splitted_output = split_words(output.first)
					if (splitted_output - splitted_input).empty? &&
splitted_input.length == splitted_output.length
						output << input
						break
					elsif outputs.last == output
						outputs << [input]
					end
				end
			else
				outputs << [input]
			end
		end
	end

	def arrange_outputs(output)
		output.sort_by{ |o| o.length }
	end

	def prepare_output
		compare
		outputs.map!{ |output| output.uniq }
		outputs.sort_by!{ |o| o.length }.reverse!
		print_output(outputs)
	end

	def print_output(output)
		output.each do |o|
			puts "#{o.length} : #{o.join(',')}"
		end
	end
end

anagram = Anagram.new
anagram.prepare_output
```

Output:

```
F:\>ruby anagram.rb
Enter words and enter blank to view result
how
who
here
paw
wap
awp

3 : paw,wap,awp
2 : how,who
1 : here

F:\>
```

2) Consider different test cases for normal and abnormal operation and inputs;

```
require './anagram.rb'
require 'pry'

describe Anagram do

 let(:inputs) { ['how','who','here','paw','wap','awp'] }

describe '#output' do

 let(:outputs) { [['paw', 'wap', 'awp'], ['how', 'who'], ['here']] }

subject { Anagram.new(inputs).output }

 it 'gives outpus in required array' do

expect(subject).to eql(outputs)

end

end

 describe '#formatted_output' do

 let(:outputs) do

 "3: awp, paw, wap \n2: how, who \n1: here \n"

 end

 subject { Anagram.new(inputs).formatted_output }

 it 'gives text in expected format' do

 expect(subject).to eql(outputs)

end

 end

end
```

```
Finished in 0.00142 seconds (files took 0.16068 seconds to load)
2 examples, 0 failures
```

3) **Use test driven development method and present the TDD process in your code environment (not the general flow) in this assignment;**

Here is a basic TDD steps that I followed to develop this code for anagrams.

Process:

First Unit Test output

```
Failures:

  1) Anagram gives outpus in correct format
     Failure/Error: expect(subject).to eql(outputs)

       expected: [["paw", "wap", "awp"], ["how", "who"], ["here"]]
            got: ["how", "who", "here", "paw", "wap", "awp"]

       (compared using eql?)
     # ./spec/anagram_spec.rb:11:in `block (2 levels) in <top (required)>'

Finished in 0.00913 seconds (files took 0.16325 seconds to load)
1 example, 1 failure

Failed examples:

rspec ./spec/anagram_spec.rb:10 # Anagram gives outpus in correct format
```

Here I found that the output is being received as an array which I gave as an input.

```
Failures:

  1) Anagram gives outpus in correct format
     Failure/Error: expect(subject).to eql(outputs)

       expected: [["paw", "wap", "awp"], ["how", "who"], ["here"]]
            got: [["how", "who"], ["how", "who"], ["here"], ["paw", "wap", "awp"], ["paw"
, "wap", "awp"], ["paw", "wap", "awp"]]

       (compared using eql?)
     # ./spec/anagram_spec.rb:11:in `block (2 levels) in <top (required)>'

Finished in 0.00933 seconds (files took 0.16166 seconds to load)
1 example, 1 failure
```

On grouping words with same character, we get an array that looks similar to what we want but difference is that there is multiple array with same values in different index.

```
Finished in 0.00095 seconds (files took 0.16029 seconds to load)
1 example, 0 failures
```

After refactoring, we get our expected array and test passed.
Now the next step is that, our outputs should be in string like

3: paw, wap, awp
2: how, who
1: here

So, converting array to string as per desired

```
Failures:

  1) Anagram#formatted_output gives text in expected format
     Failure/Error: expect(subject).to eql(outputs)

       expected: "3: awp, paw, wap \n 2: how, who \n 1: here"
            got: nil

       (compared using eql?)
     # ./spec/anagram_spec.rb:22:in `block (3 levels) in <top (required)>'

Finished in 0.02945 seconds (files took 0.56228 seconds to load)
2 examples, 1 failure
```

We defined a method to return desired string and test failed since method return nil value.

After refactoring and running the test, we passed our test.

```
Finished in 0.00142 seconds (files took 0.16068 seconds to load)
2 examples, 0 failures
```

Q3. What are the benefits of using change request form as the central document in the change management process?

A change management process can be broadly understood as a chain of activities that is adopted by the change management team in order to deal with the modifications in a project. The benefit of this process is that it reduces the resistance to the change on the course of implementing the organizational change. Preceding the change management process, the development submits a change request. This change request can be comprehended as a call for alteration in the system. A change request is called whenever a team member trusts that a change is required to achieve the objective. This kind of change request is normally submitted using a form which is commonly known as a change request form. This form contains all the information related to the required changes along with its impacts and the consequences. In addition, this form consists of the facts that supports for the introduction of a change. Overall, a change request form can be considered as a central document in the change management process.

The major benefits of using a change request form as the central document in the change management process can be listed as:

a. It maintains a track of all the requestion changes so that it can be used for future reference.
b. It assists in the identification of most probable changes prior to their occurrence.
c. It helps in conducting a cost-benefit analysis of the requested change.
d. It describes the impact and the consequences of implementing the change.
e. It generates a priority list of the requested changes on the basis of their urgency.
f. It helps to identify all the prerequisites for implementing the requested change.
g. It monitors and controls the impact of the implemented changes.
h. It aids in controlling the extent of change approval.

Lack of a proper change control might lead to the time and cost overruns thereby resulting in the delay of the project completion. In this case, a better implemented change request form could help by monitoring and controlling the changes which, in turn, increases the probability of achievement of project objective.