Software Measurement (SOEN6611)
Summer 2018
Descriptive Statistics
Team J
Deliverable 1

Ksenia Popova, Farhan Shaheen, Himalaya Reddiboina,
Kaushik Samanta, Adrien Poupa, Yousef Saatchi Tabriz

Contents

1 Goal-Question-Metric

1.1 Goal

To create a subsystem, integrated in a Student Information System (SIS) in order to be used to collect statistics about students grades. The subsystem is named "Descriptive Statistics". The subsystem must be accessible by the SIS and usable by the students, the administration and the professors. Descriptive Statistics has to provide students statistics of their grades in an easy manner. Professors must be able to enter the grades and see the students' statistics quickly. The administrators must be able to oversee the system without any hassle. Descriptive Statistics must be able to recover from errors.

Note: in this document, "users" designate the actors of Descriptive Statistics; professors, students and a department member.

1.2 Question-Metric

1. (a) Question: How to ensure that the Descriptive Statistics is connected to the SIS?
   (b) Metric: Create Integration tests and assess their success to evaluate the connection between Descriptive Statistics and the SIS. This is an objective metric, based on the viewpoint on the subsystem.
   (c) Mechanism:
      i. Owner = Lead Developer
      ii. Frequency Collected = Following Each Major Software Release
      iii. Frequency Reported = Quarterly
2. (a) Question: How to ensure the system is easy to use by the students?
   (b) Metric: Conduct a survey using the System Usability Scale (Mifsud, 2015). This is a subjective metric, based on the viewpoint of the users.
   (c) Mechanism:
      i. Owner = Product Manager

ii. Frequency Collected = Following The End of the First Session
iii. Frequency Reported = Quarterly

3. (a) Question: How to ensure the system is easy to use by the professors?
   (b) Metric: Conduct a survey using the Computer System Usability (Mifsud, 2015). This is a subjective metric, based on the viewpoint of the users.
   (c) Mechanism:
      i. Owner = Product Manager
      ii. Frequency Collected = Following The End of the First Session
      iii. Frequency Reported = Quarterly
4. (a) Question: How to ensure the system is easy to use by the department?
   (b) Metric: Conduct a survey using the Computer System Usability (Mifsud, 2015) at the end of the first session. This is a subjective metric, based on the viewpoint of the users.
   (c) Mechanism:
      i. Owner = Product Manager
      ii. Frequency Collected = Following The End of the First Session
      iii. Frequency Reported = Quarterly
5. (a) Question: How to ensure that a professor can enter a grade in Descriptive Statistics in no more than 5 seconds?
   (b) Metric: Calculate the Time-Based Efficiency (Mifsud, 2015). This is an objective metric, based on the viewpoint on the subsystem.
   (c) Mechanism:
      i. Owner = Lead Developer
      ii. Frequency Collected = Every Day
      iii. Frequency Reported = Quarterly
6. (a) Question: How to ensure that professors, students and department does not give up on using Descriptive Statistics?

(b) Metric: Calculate the Completion Rate (Sauro, 2011). This is an objective metric, based on the viewpoint on the subsystem.

(c) Mechanism:

i. Owner = Product Manager

ii. Frequency Collected = Every Day

iii. Frequency Reported = Quarterly

7. (a) Question: How to prevent errors in Descriptive Statistics?

(b) Metric: Record every error encountered by users in Descriptive Statistics (Seeley, 2010). This is an objective metric, based on the viewpoint on the subsystem.

(c) Mechanism:

i. Owner = Lead Developer

ii. Frequency Collected = Every Day

iii. Frequency Reported = Quarterly

8. (a) Question: How to ensure that Descriptive Statistics meet students', professors' and department's expectations?

(b) Metric: Use the Single Ease Question metric (Mifsud, 2015) before and after a task to see the difference. in terms of expectations. This is a subjective metric, based on the viewpoint of the users.

(c) Mechanism:

i. Owner = Product Manager

ii. Frequency Collected = Before and After Every Task

iii. Frequency Reported = Quarterly

9. (a) Question: How to ensure that Descriptive Statistics does not take more than one second for a calculation to complete?

(b) Metric: Log Load Time (West, 2015). This is an objective metric, based on the viewpoint on the subsystem.

(c) Mechanism:

i. Owner = Lead Developer

ii. Frequency Collected = Every Day

iii. Frequency Reported = Quarterly

10. (a) Question: How to make sure that users are able to use the system autonomously?

(b) Metric: Record the Frequency of help and documentation use. This is an objective metric, based on the viewpoint on the subsystem and the support staff.

(c) Mechanism:

i. Owner = Product Manager

ii. Frequency Collected = Every Day

iii. Frequency Reported = Quarterly

11. (a) Question: How many clicks does Descriptive Statistics require to complete the tasks?

(b) Metric: Record the Number of Clicks (Kelkar, 2017). This is an objective metric, based on the viewpoint on the subsystem.

(c) Mechanism:

i. Owner = Product Manager

ii. Frequency Collected = Every Day

iii. Frequency Reported = Quarterly

12. (a) Question: How to make sure that the learning curve of Descriptive Statistics is smooth?

(b) Metric: Compare the Time-Based Efficiency (Mifsud, 2015) for a first time user and a returning user. This is an objective metric, based on the viewpoint on the subsystem.

(c) Mechanism:

i. Owner = Product Manager

ii. Frequency Collected = Every Day

iii. Frequency Reported = Quarterly

2 Use Case Model

2.1 Use Case Diagram

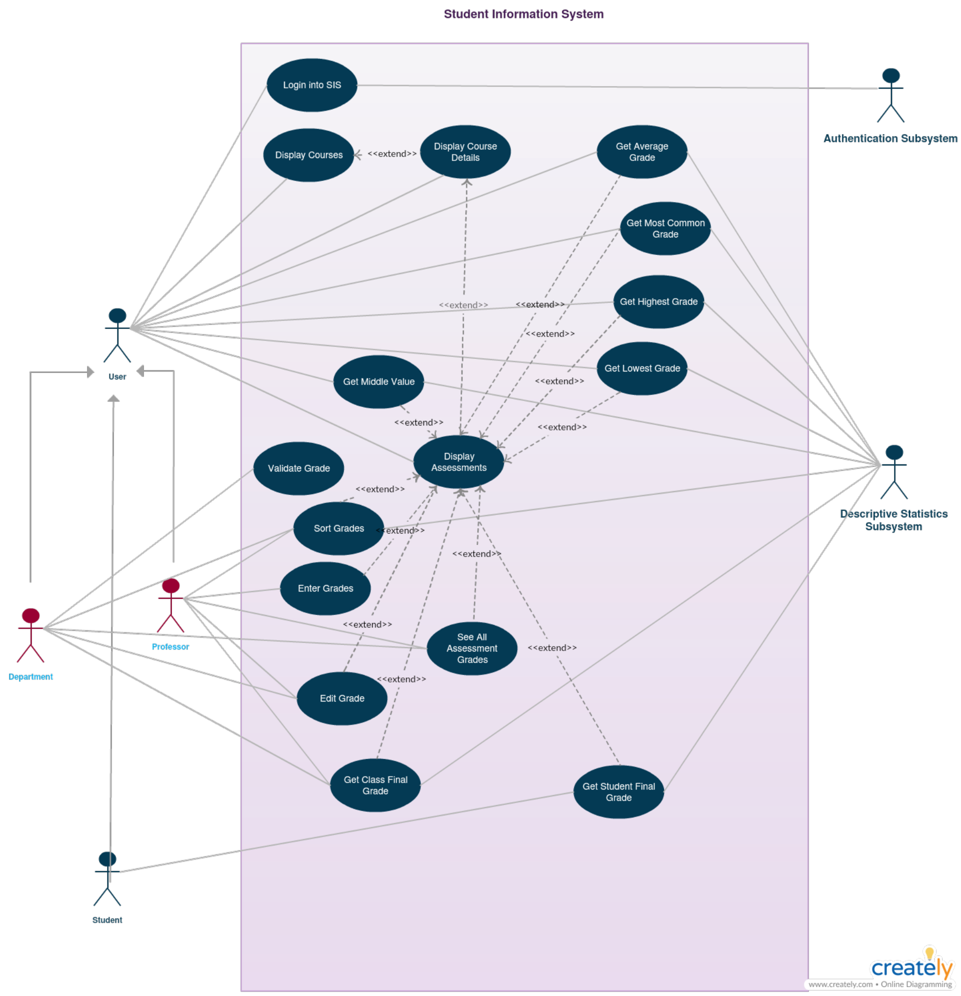

Figure 1: Use Case Model

2.2 Use Cases

Use Case ID	UC-1
Use Case Name	Log in to the System
Primary Actors	• Department • Professor • Student
Secondary Actor	Authentication Subsystem
Priority	High
Description	User can login into the System.
Pre-conditions	• User has a valid account.
Post-conditions	• User logged in.
Normal Flow	1. User requests the login page of the system. 2. System displays the login page. 3. User enters their username and their password. 4. User clicks on “Login” . 5. System checks the User’s credentials. 6. System redirects User to the homepage. 7. System displays the homepage. 8. User sees the homepage.
Alternate Flow	5. a. Credentials provided by the User were invalid.

Use Case ID	UC-2
Use Case Name	Display Courses
Primary Actors	• Department • Professor • Student
Secondary Actor	None
Priority	High
Description	User can display a list of their courses.
Pre-conditions	• User is logged in to the System.
Post-conditions	• A list of courses were displayed.
Normal Flow	1. User requests the list of the courses available. 2. The System displays the list of the courses available: 3. If the user is a Professor: list of courses taught by the Professor. 4. If the user is a Department member: list of the courses that the Department offers. 5. If the user is a Student: list of courses that the Student is enrolled in.
Alternate Flow	2. a. Professor is teaching a course but does not have access to the course. 2. b. Department offers a course but does not have access to the course. 2. c. Student is enrolled in a course but does not have access to the course.

Use Case ID	UC-3
Use Case Name	Display Course Details
Primary Actors	• Department • Professor • Student
Secondary Actor	None
Priority	High
Description	User can display a list of their courses.
Pre-conditions	• User is logged in to the System. • User requested to see course details from the course list. • Professor has access to the course, course is contained in Department, or Student has access to the course.
Post-conditions	• A course was displayed.
Normal Flow	1. User selects a course from the course list. 2. The System displays the course details: 3. Course code 4. Course name 5. Professor of the course 6. Department of the course 7. Number of Students enrolled in the course 8. Link to “Assessments” 9. Link to “All Assessments” if the user is a Professor or a Department member
Alternate Flow	

<table>
<tr><td>Use Case ID</td><td>UC-4</td></tr>
<tr><td>Use Case Name</td><td>Display Assessments</td></tr>
<tr><td>Primary Actors</td><td>Department
Professor
Student</td></tr>
<tr><td>Secondary Actor</td><td>None</td></tr>
<tr><td>Priority</td><td>High</td></tr>
<tr><td>Description</td><td>User can display assessments for a course.</td></tr>
<tr><td>Pre-conditions</td><td>User is logged in to the System.
User requested to see assessments for a course.</td></tr>
<tr><td>Post-conditions</td><td>Assessments were displayed.</td></tr>
<tr><td>Normal Flow</td><td>User selects clicks on “Assessments” in a course details.
The System displays the assessments links:
Link to “Lowest Grade”
Link to “Highest Grade”
Link to “Average Grade”
Link to “Most Common Grade”
Link to “Middle Grade”
Link to “Final Grade” if the user is a Student
Link to “Final Grades” if the user is a Professor or a Department member.
Link to “Enter grades” if the user is a Professor
Link to “Edit grades” if the user is a Professor</td></tr>
<tr><td>Alternate Flow</td><td></td></tr>
</table>

Use Case ID	UC-5
Use Case Name	Get Lowest Grade
Primary Actors	• Department • Professor • Student
Secondary Actor	Descriptive Statistics Subsystem
Priority	High
Description	User can see the lowest grade of an assessment.
Pre-conditions	• User is logged into the System. • The grades are entered in the System. • User requested to see assessments for a course.
Post-conditions	• Lowest grade is displayed.
Normal Flow	1. User clicks on "Lowest grade" . 2. The System displays the lowest grade.
Alternate Flow	

Use Case ID	UC-6
Use Case Name	Get Highest Grade
Primary Actors	• Department • Professor • Student
Secondary Actor	Descriptive Statistics Subsystem
Priority	High
Description	User can see the highest grade of an assessment.
Pre-conditions	• User is logged into the System. • The grades are entered in the System. • User requested to see assessments for a course.
Post-conditions	• Highest grade is displayed.
Normal Flow	1. User clicks on “Highest grade” . 2. The System displays the highest grade.
Alternate Flow	

Use Case ID	UC-7
Use Case Name	Get Average Grade
Primary Actors	• Department • Professor • Student
Secondary Actor	Descriptive Statistics Subsystem
Priority	High
Description	The Department and Professors can see the Average grade of an assessment.
Pre-conditions	• User is logged in into the System. • The grades are entered in the System. • User requested to see assessments for a course.
Post-conditions	• Average grade is displayed
Normal Flow	1. User clicks on “Average grade” . 2. The System displays the average grade.
Alternate Flow	

Use Case ID	UC-8
Use Case Name	Get Most Common Grade
Primary Actors	• Department • Professor • Student
Secondary Actor	Descriptive Statistics Subsystem
Priority	High
Description	The Department and Professors can see the common grade of assessment.
Pre-conditions	• User is logged into the System. • The grades are entered in the System. • User requested to see assessments for a course.
Post-conditions	• Most common grade is displayed
Normal Flow	1. User clicks on "Most common grade" . 2. The System displays the most common grade.
Alternate Flow	

Use Case ID	UC-9
Use Case Name	Get Middle Value
Primary Actors	• Department • Professor • Student
Secondary Actor	Descriptive Statistics Subsystem
Priority	High
Description	The Department and Professors can see the middle grade of assessment.
Pre-conditions	• User is logged in into the System. • The grades are entered in the System. • Professor has access to the course, course is contained in Department, or Student has access to the course • User requested to see assessments for a course.
Post-conditions	• Middle grade is displayed.
Normal Flow	1. User clicks on “Middle grade” . 2. The System displays the middle grade.
Alternate Flow	

Use Case ID	UC-10
Use Case Name	Get Class Final Grade
Primary Actors	• Department • Professor
Secondary Actor	Descriptive Statistics Subsystem
Priority	High
Description	The Department and Professors can see the final grade of the class.
Pre-conditions	• User is logged into the System. • The grades are entered in the System. • The course is complete. • User requested to see assessments for a course.
Post-conditions	• Final grade of the class is displayed.
Normal Flow	1. User clicks on "Final grades" . 2. The System displays the Final grade for every Student. Student above the average score as pass and student below 20% of average are fail. This is computed using the standard deviation.
Alternate Flow	4. a. Professor has not submitted the grades by the time students check for their grades. 4. b. Professor has submitted the grades but the department has not accepted the grades by the time students check for their grades.

Use Case ID	UC-11
Use Case Name	Get Student Final Grade
Primary Actors	Student
Secondary Actor	Descriptive Statistics Subsystem
Priority	High
Description	Students can see their own final grade of assessment.
Pre-conditions	• User is logged in into the System. • The grades are entered in the System • Student has access to the course. • The course is complete. • User requested to see assessments for a course.
Post-conditions	• Final grade of the Student was displayed.
Normal Flow	1. User clicks on "Final grade" . 2. The System displays the Final grade for the Student. Student above the average score as pass and student below 20% of average are fail. This is computed using the standard deviation.
Alternate Flow	

Use Case ID	UC-12
Use Case Name	Sort Grades
Primary Actors	• Department • Professors
Secondary Actor	Descriptive Statistics Subsystem
Priority	Medium
Description	The Department and Professors can see the final grade of the class in both ascending and descending order.
Pre-conditions	• User is logged in into the System. • The grades are entered in the System. • User requested to see assessments for a course.
Post-conditions	• Final grade of the class in both ascending and descending order of the class was displayed.
Normal Flow	1. User clicks on "Final grades" . 2. The System displays the Final grade for every student (Student above the average score as pass and student below 20% of average are fail). 3. User selects the "Sort" button 4. Final grades are displayed in ascending when clicked once, displayed in descending order when clicked twice.
Alternate Flow	

Use Case ID	UC-13
Use Case Name	Enter Grades
Primary Actor	Professor
Secondary Actors	None
Priority	High
Description	Professor enters grades of Students for a specific course.
Pre-conditions	• Professor has access to the course • Professor requested to see assessments.
Post-conditions	Grades are stored in the system.
Normal Flow	1. Professor clicks on "Enter grades" . 2. Professor selects a student. 3. Portal asks the grade of the student. 4. Professor enters the grade. 5. Portal stores the grade in the database.
Alternate Flow	1. a. No assessment exists for the course, the portal shows a message indicating the course has no assessments posted yet. 1. a. The selected assessment is already graded. The portal shows a message indicating that the grades can only be edited. 5. a. The entered grade is not valid. The portal asks the professor to enter a valid grade.

Use Case ID	UC-14
Use Case Name	Edit Grades
Primary Actor	• Professor • Department
Secondary Actors	None
Priority	High
Description	User can edit the grades of Students for a specific course.
Pre-conditions	• User is logged into the System. • User requested to see a course.
Post-conditions	Grades are stored in the System.
Normal Flow	1. Professor clicks on "Edit grades" . 2. User selects a student. 3. User edits the grade. 4. Portal stores the grade in the database.
Alternate Flow	1. a. No assessment exists for the course, the portal shows a message indicating the course has no assessments posted yet. 4. a. The entered grade is not valid. The portal asks the professor to enter a valid grade.

Use Case ID	UC-15
Use Case Name	See All Assessments Grades
Primary Actor	• Professor • Department
Secondary Actors	None
Priority	High
Description	User can see all the assessments of a class.
Pre-conditions	• User is logged into the System. • The grades has been submitted into the System by the Professor. • User requested to see a course
Post-conditions	• Grades were displayed without any mistake.
Normal Flow	1. User clicks on "All assessments" button. 2. System displays all assessments of all students for the course.
Alternate Flow	2. a. No assessment has been submitted yet.

Use Case ID	UC-16
Use Case Name	Validate Grades
Primary Actor	Department member
Secondary Actor	None
Priority	Medium
Description	Department member can validate the grades.
Pre-conditions	• The grades are entered in the System. • The course is contained in Department. • Department user is logged into the System
Post-conditions	• The grades were validated.
Normal Flow	1. Department member requests the list of the courses having grades waiting to be validated. 2. The System displays the list of the courses having grades waiting to be validated. 3. Department member selects the course. 4. The System displays the grades waiting to be validated. 5. Department member clicks on "Validate" to validate the grades. 6. The System displays a confirmation message. 7. The System redirects the Department member to the list of the courses having grades waiting to be validated.
Alternate Flow	1. a. Department contains a course but does not have access to the course. 5. a. Department member clicks on "Refuse" to refuse the grades, asking for the Professor to review them before re-submitting them.

3 Use Case Points and COCOMO

3.1 Use Case Points

In this part, we will calculate the effort estimate using the Use Case Points (UCP) approach. Let the Productivity Factor (PF) be 20. The formulas used to calculate the effort are as follows:

$$Effortestimate = UCP * PF$$

and

$$UCP = UUCP * TCF * ECF$$

The Unadjusted Use Case Points (UUCP) is calculated using the Unadjusted Actor Weight (UAW) and Unadjusted Use Case Weight (UUCW):

$$UUCP = UAW + UUCW$$

Now, we will calculate the UAW using the following table:

Table 1: Actor Weights

Actor	Type	Weight
Professor	Complex Actor	3
Student	Complex Actor	3
Department Member	Complex Actor	3
Authorization Subsystem	Simple Actor	1
Descriptive-Statistics Subsystem	Simple Actor	1

As one can see, our system contains six actors: the human actors are complex while the subsystems are simple. Therefore,

$$UAW = 3 * 3 + 2 * 1 = 11$$

Then, we calculate UUCW.

Table 2: Number of Transactions per Use Case and UUCW

Use Case Number	Number of Transactions	UUCW
1	2	5
2	1	5
3	1	5
4	1	5
5	1	5
6	1	5
7	1	5
8	1	5
9	1	5
10	1	5
11	1	5
12	2	5
13	2	5
14	1	5
15	1	5
16	3	5

Therefore, we have:

$$UUCW = 16 * 5 = 80$$

Now, UUCP can be calculated:

$$UUCP = UAW + UUCW$$

$$UUCP = 11 + 80 = 91$$

We will now focus on Technical Complexity Factor (TCF) calculation. Below is a table showing all the factors and their weight for the project:

Table 3: TCF calculation

TCF	Description	Weight	Factor	Weight * Factor
1	Distributed System	2	3	6
2	Performance	1	4	4
3	End User Efficiency	1	4	4
4	Complex Internal Processing	1	2	2
5	Reusability	1	2	2
6	Easy to Install	0.5	1	0.5
7	Easy to Use	0.5	5	2.5
8	Portability	2	2	4
9	Easy to Change	1	2	2
10	Concurrency	1	4	4
11	Special Security Features	1	3	3
12	Provides Direct Access for Third Parties	1	1	1
13	Special User Training Facilities are Required	1	0	0

For formatting purposes, Weight means Technical Complexity Factor Weight (WTi) and Factor means Perceived Impact Factor (Fi).

From this table, TCF can be calculated by computing the sum of the WTi and the Fi:

$$TCF = C_1 + C_2 \sum_{i=1}^{13} (W_{Ti} * F_i)$$

$$TCF = 0.6 + 0.01 * (6 + 4 + 4 + 2 + 2 + 0.5 + 2.5 + 4 + 2 + 4 + 3 + 1 + 0)$$

$$TCF = 0.95$$

We will now focus on Environment Complexity Factor (ECF) calculation. Below is a table showing all the factors and their weight for the project:

Table 4: ECF calculation

ECF	Description	Weight	Factor	Weight * Factor
1	Familiarity with Use Case Domain	1.5	5	7.5
2	Part-Time Workers	-1	1	-1
3	Analyst Capability	0.5	3	1.5
4	Application Experience	0.5	2	1
5	Object-Oriented Experience	1	3	3
6	Motivation	1	4	4
7	Difficult Programming Language	-1	1	-1
8	Stable Requirements	2	4	8

For formatting purposes, Weight means Environmental Complexity Factor Weight (WEi) and Factor means Fi.

From this table, TCF can be calculated by computing the sum of the WEi and the Fi:

$$ECF = C_1 + C_2 \sum_{i=1}^{13} (W_{Ei} * F_i)$$

$$ECF = 1.4 - 0.03 * (7.5 - 1 + 1.5 + 1 + 3 + 4 - 1 + 8)$$

$$ECF = 0.71$$

We can now calculate UCP:

$$UCP = UUCP * TCF * ECF$$

$$UCP = 91 * 0.95 * 0.71$$

$$UCP = 61.3795$$

Therefore, the effort estimate is:

$$Effort = UCP * PF$$

$$Effort = 61.3795 * 20$$

$$Effort = 1227.59$$

We can conclude that the effort estimate is of 1228 person-hours.

3.2 COCOMO

In this part, we will calculate the Constructive Cost Model (COCOMO) effort estimation using the Basic COCOMO approach.
The equation for effort estimation is as follows:

$$E = a * S^b * F$$

Descriptive-Statistics can be undertaken as an organic project, therefore:

$$a = 2.4$$

$$b = 1.05$$

$$F = 1$$

The value of S is coming from later in this document, in the Physical Source Line of Code (SLOC) paragraph. We have counted 184 SLOC using the physical model, which gives:

$$S = 0.193$$

This results in:

$$E = 2.4 * 0.193^{1,05} * 1$$

$$E = 0.42662511285$$

COCOMO gives an effort estimate of 0.42 person-months.
Assuming the team works 20 days per month, 8 hours each day, then converting E in person-hours would give:

$$E = 0.42662511285 * 20 * 8 = 68.260018056$$

Therefore, the effort estimate is about $E = 69$ person-hours using the Basic COCOMO approach.

The difference is coming from the fact that our calculation of UCP is done using the use cases that are closely related to the Descriptive-Statistics subsystem whereas the Basic COCOMO approach is based on the actual SLOC we have written. Those lines contain only the Descriptive-Statistics subsystem and none of the interaction described in the Use Case Model (UCM). As a consequence, the estimation given by the UCP seems to be more accurate.

4 Coding and Testing

Main programming language used to implement the descriptive statistics for the project is R Language.

4.1 Coding

We have five main classes in the project:

```
data_provider <- setRefClass("data_provider",
ds_list <- setRefClass("ds_list",
descriptive_statistics <- setRefClass("descriptive_statistics",
math_util <- setRefClass("math_util",
driver <- setRefClass("driver",
```

4.1.1 Class "data_provider" is responsible for data input

- Reading data from file:

We are using a CSV file as input which has 1000 random number that range between 1 to 100.

List is also being sort after data has been retrieved from the file. This is done for optimal performance.

- Code Snippet:

```
data_provider <- setRefClass("data_provider",
fields = list(grades = "ds_list"),
methods = list(
    get_random_numbers = function(){
        list <- read.csv("data.csv",header=F)$V1

        grades <<- ds_list(items = list)
        grades$sort()

        return (grades)
    }
)
)
```

4.1.2 Class "ds_list" has sorting functionality and also sorts the list

- Sorting:

We are using Radix sort to sort the list.

Because of the nature of the input, we can exploit the radix sort to our advantage. Since the maximum number in the list can be 100 regardless of the length of the list, we are able to sort the data in $\Theta(dn)$. $d = 3$ is our case.

That is a huge performance improvement as we are able to sort the list linearly.

- Code Snippet:

```
getmax = function(list, n){
    max = list[[1]]
    for(i in 2:n){
        if(list[[i]] > max)
            max = list[[i]];
    }
    return (max)
},

countsort = function(list, n, digitno){
    outputlist <- c(1:n)
    i <- 0
    count <- c(0,0,0,0,0,0,0,0,0,0)

    # count didgit occurences
    for(i in 1:n){
        j <- ((list[[i]]/digitno) %% 10) + 1
        count[[j]] <- count[[j]] + 1
    }

    for(i in 2:10){
        count[[i]] <- count[[i]] + count[[i-1]]
    }

    for(i in n:1){
        k <- ((list[[i]]/digitno) %% 10) + 1
```

```
        l <- count[[k]]
        outputlist[[l]] <- list[[i]]
        count[[k]] <- count[[k]] - 1
    }
    return (outputlist)
},

sort = function(){
    # find maximun number from the list
    item_length = length(items)
    max = getmax(items, item_length)
    j <- 1
    for(digitno in 1:(max/j)){
        items <<- countsort(items, item_length, digitno)
        j <- j + 1
    }
}
```

4.1.3 Class "descriptive_statistics" has all the descriptive statistics functions

There are 6 functions as follows.

1. (m) minimum
2. (M) maximum
3. (o) mode
4. (d) median
5. (μ) arithmetic mean
6. (σ) standard deviation

4.1.3.1 (m) minimum

The **minimum**, m, is the smallest of the values in the given data set. (m need not be unique.)

Function "min_grade" finds the smallest number from the list of n numbers.

Since list was sorted in the beginning. 1st item in the list is the minimum. Asymptotic complexity for calculating (m) is $O(1)$, which is constant.

- Code Snippet:

```
min_grade = function() {
    return (grades$items[[1]])
}
```

4.1.3.2 (M) maximum

The **maximum**, M, is the largest of the values in the given data set. (M need not be unique.)

Function "max_grade" finds the largest number from the list of n numbers.

Since list was sorted in the beginning. Last item in the list is the maximum. Asymptotic complexity for calculating (M) is $O(1)$, which is constant.

- Code Snippet:

```
max_grade = function() {
    return (grades$items[[grades$size()]])
}
```

4.1.3.3 (o) mode

The **mode**, o, is the value that appears most frequently in the given data set. (o need not be unique.)

There can be more than one mode.

Since list was sorted in the beginning. Asymptotic complexity for calculating (o) is $O(n)$, which is linear.

- Code Snippet:

```
grades_mode = function() {
```

```
    n <- grades$size()
    max <- 1
    currentAmount <- 1
    n <- n-1
    modeRes <- c()
    mode_size <- 0
    uniq_amount <- 1

    for (i in c(1:n)) {
      if (grades$items[i] == grades$items[i + 1]) {
        currentAmount <- currentAmount+1
      }
      else {
        if(currentAmount == max) {
          modeRes <- c(modeRes, grades$items[i])
          mode_size <- mode_size+1
        }
        if (currentAmount > max) {
          max <- currentAmount
          modeRes <- grades$items[i]
          mode_size <- 1
        }
        currentAmount <- 1
        uniq_amount <- uniq_amount+1
      }
    }

    if (currentAmount == max) {
      modeRes <- c(modeRes, grades$items[n+1])
      mode_size <- mode_size+1
    }

    if (currentAmount > max) {
      modeRes <- grades$items[i]
      mode_size <- 1
    }

    if (mode_size == uniq_amount)
      modeRes <- c("NA")

    return (modeRes)
}
```

4.1.3.4 (d) median

The **median**, d, is the **middle number if n is odd**, and is the **arithmetic mean of the two middle numbers if n is even.**

Since list was sorted in the beginning. Asymptotic complexity for calculating (d) is $O(1)$, which is constant.

- Code Snippet:

```
grades_median = function() {
    size <- grades$size()

    if(size %% 2 == 1) {
      value <- grades$items[[size / 2 + 1]]
    }
    else{
      value <- (grades$items[[size / 2]] + grades$items[[size / 2 +
          1 ]]) / 2
    }
    return (value)
}
```

4.1.3.5 (μ) arithmetic mean

The **arithmetic mean**, μ, is given by

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Asymptotic complexity for calculating (μ) is $O(n)$, which is linear.

- Code Snippet:

```
grades_mean = function() {
    counter <- 0

    size <- grades$size()

    for(i in 1:size){
      counter <- counter + grades$items[[i]]
```

```
    }

    value <- counter/size

    return (value)
}
```

4.1.3.6 (σ) standard deviation

The **standard deviation**, σ, is given by

This Standard deviation (SD) function is population standard deviation.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

Asymptotic complexity for calculating (σ) is: $\Theta(n)$ for mean + $\Theta(n)$ for sum_custom() + $\Theta(\log n)$ for square_root; that is $O(n)$, which is linear.

- Code Snippet:

```
grades_std_dev = function() {
    math <- math_util()

    mean <- grades_mean()

    std_dev <- math$square_root(sum_custom((grades$items - mean)^2)
        / grades$size())

    return(std_dev)
}
```

4.1.4 Class "math_util"

This class has all the math functions implementations which can be used by descriptive statistics functions.

In out project we implemented square root function.

- Code Snippet:

```
math_util <- setRefClass("math_util",
methods = list(
    square_root = function(num) {
        low = 0
        high = num
        mid = 0

        while (high - low > 0.0000001) {
            mid <- low + (high - low) / 2
            if (mid*mid > num)
                high <- mid
            else
                low <- mid
        }
        return(mid)
    }
)
)
```

4.1.5 Class Driver

This is the main driver class which is used to run all the descriptive functions.

- Code Snippet:

```
#R version 3.3.2
#Author: Group J
#SOEN 6611 Project
#Driver
#May-June 2018

source("descriptive_statistics.r")
source("data_provider.r")

driver <- setRefClass("driver",
     fields = list(dp = "data_provider", ds =
         "descriptive_statistics"),
     methods = list(
       constructor = function(){
         dp <<- data_provider()
```

```
        random_grades <- dp$get_random_numbers()

        ds <<- descriptive_statistics(grades = random_grades)

        max_grade <- ds$max_grade()
        min_grade <- ds$min_grade()
        grades_mean <- ds$grades_mean()
        grades_median <- ds$grades_median()
        grades_mode <- ds$grades_mode()

        #standard deviation
        std_dev <- ds$grades_std_dev()

        print(paste0("maximum grade: ", max_grade))
        print(paste0("minimum grade: ", min_grade))
        print(paste0("grades mean: ", grades_mean))
        print(paste0("grades median: ", grades_median))
        print(paste0("grades mode: ", grades_mode))
        print(paste0("grades standard deviation: ", std_dev))
      }
    )
)

main<-driver()
main$constructor()
```

4.2 Testing

4.2.1 Running the code:

We are using driver to run all the descriptive statistics functions. Code given in section 4.1.5

- Output:

```
[1] "maximum grade: 100"
[1] "minimum grade: 1"
[1] "grades mean: 47.324"
[1] "grades median: 46"
[1] "grades mode: 22"
[1] "grades standard deviation: 28.7191055402808"
```

4.2.2 Unit testing in R:

Using assertthat in R language.

assertthat returns TURE or FALSE which can be used to check if the intended function is working properly.
- Code Snippet:

```r

library("assertthat")

source("descriptive_statistics.r")
source("data_provider.r")

dp <- data_provider()
random_grades <- dp$get_random_numbers()

ds <- descriptive_statistics(grades = random_grades)

max_grade <- ds$max_grade()
min_grade <- ds$min_grade()
grades_mean <- ds$grades_mean()
grades_median <- ds$grades_median()
grades_mode <- ds$grades_mode()

#standard deviation
std_dev <- ds$grades_std_dev()

#
    https://www.calculatorsoup.com/calculators/statistics/mean-median-mode.php
are_equal(max_grade, 100)
are_equal(min_grade, 1)
are_equal(grades_mean, 47.324)
are_equal(grades_median, 46)
are_equal(grades_mode, c(22))

# http://www.calculator.net/standard-deviation-calculator.html
are_equal(std_dev, 28.719105557103)

# R compare with R built-in functions
max(random_grades$items)
min(random_grades$items)
mean(random_grades$items)
median(random_grades$items)
```

```r

# data <- c(93, 45, 75, 96, 80, 45, 2, 66, 75)
# Test function only
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

getmode(random_grades$items)
sd(random_grades$items)

are_equal(max_grade, max(random_grades$items))
are_equal(min_grade, min(random_grades$items))
are_equal(grades_mean, mean(random_grades$items))
are_equal(grades_median, median(random_grades$items))
are_equal(grades_mode, getmode(random_grades$items))
```

- Output:

```

> are_equal(max_grade, max(random_grades$items))
[1] TRUE
> are_equal(min_grade, min(random_grades$items))
[1] TRUE
> are_equal(grades_mean, mean(random_grades$items))
[1] TRUE
> are_equal(grades_median, median(random_grades$items))
[1] TRUE
> are_equal(grades_mode, getmode(random_grades$items))
[1] TRUE
```

5 Conclusion

5.1 Cyclomatic Number

The Cyclomatic Number can be calculated using one of the following formulas:

$$v(G) = e - n + 2$$

where e represents the number of arcs of the control flow graph of the function that is studied and n the number of nodes of this graph; or using

$$v(G) = d + 1$$

where d is the number of predicates having an out-degree of 2.

For the calculation of the complexity number of the methods that we have written, we first drew the control flow graph of each function, calculated the complexity number using the first formula, then we confirmed our results using the second formula with the number of predicates.

Method	d	v(G)
max_grade	2	3
min_grade	2	3
grades_median	1	2
grades_mean	1	2
sum_custom	1	2
grades_std_dev	0	1
grades_mode	7	8
getmax	2	3
countsort	3	4
sort	1	2
size	0	1
get_random_numbers	0	1
square_root	2	3

Table 5: Cyclomatic number for Descriptive-Statistics methods

5.2 Qualitative Conclusions

All of the methods fall within the 1-10 range of cyclomatic number meaning that the risk assessment is low. The control flow graph associated to those functions is "simple". Therefore, the internal complexity of source code is low.

6 Calculations

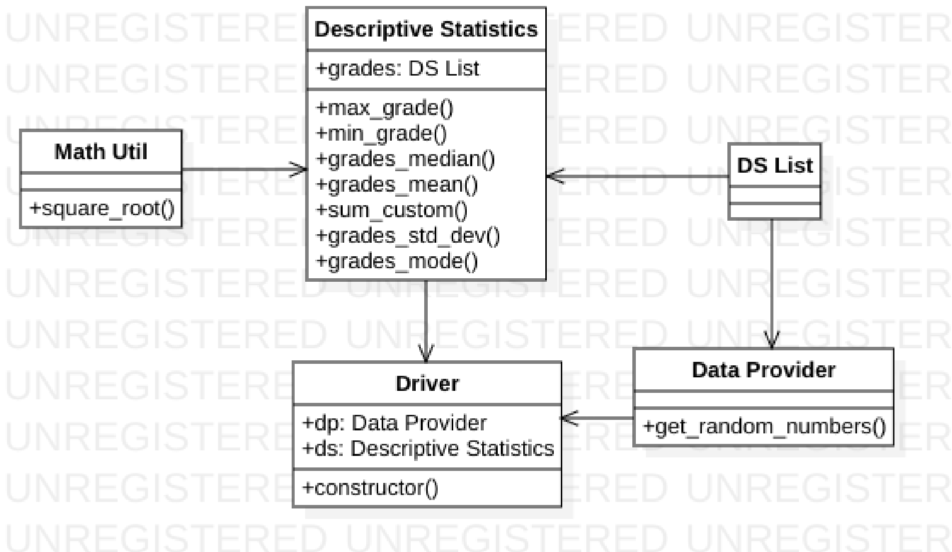

Figure 2: Class Diagram

6.1 Weighted Methods per Class

We can calculate the Weighted Methods per Class (WMC) with the following formula:

$$WMC = \sum_{i=1}^{n} C_i(M_i)$$

where C_i is the complexity of the method M_i.

We used the cyclomatic number as a measure of complexity for each method, given that weights are not normalized. Then, using the results found in the section 5, we have:

$$WMC = 1 * 3 + 2 * 4 + 3 * 4 + 4 * 1 + 8 * 1$$
$$WMC = 35$$

Therefore, the WMC is of 35 for Descriptive Statistics.

6.2 Coupling Factor

We can calculate the Coupling Factor (CF) using the following formula:

$$CF = \frac{\sum_{i=1}^{n}\sum_{i=1}^{n} IsClient(C_i, C_j)}{n^2 - n}$$

$IsClient(C_i, C_j)$ means that the class C_i has a relationship, either in the form of an attribute or a reference, with the class C_j, that is not inheritance.

Based on the diagram above, we have:

$$CF = \frac{1+1+2+1}{5^2-5}$$
$$CF = 0.25$$

Therefore, the Coupling Factor for Descriptive Statistics is equal to 0.25.

6.3 Lack of Cohesion in Methods

We can calculate the Lack of Cohesion in Methods (LCOM*) using the following formula:

$$LCOM^* = \frac{\frac{1}{a}\sum_{i=1}^{a} \mu(A_i) - m}{1 - m}$$

where a is the number of attributes, m the number of methods and $\mu(A_k)$ is the number of methods that access an attribute A_k.

7 LOC

7.1 Physical SLOC

7.1.1 Manual counting

7.1.2 Using a tool

7.2 Logical SLOC

7.2.1 Manual counting

8 Analysis: Correlations between the data for Logical SLOC and WMC

8.1 Using Scatter Plot

8.2 Using a Correlation Coefficient

Glossary

API Application Programming Interface. 1

CF Coupling Factor. 1, 44

COCOMO Constructive Cost Model. 1, 29

Cyclomatic Number Software metric, used to indicate the complexity of a program. 1

ECF Environment Complexity Factor. 1, 28

Fi Perceived Impact Factor. 1, 27, 28

GQM Goal Question Metric. 1

Integration tests tests performed to detect defaults in the interation between integrated components or systems. 1, 4

KDSI Kilo Delivered Source Instructions. 1

LCOM* Lack of Cohesion in Methods. 1, 44

PF Productivity Factor. 1, 25

SD Standard deviation. 1, 36

SIS Student Information System. 1, 4

SLOC Source Line of Code. 1, 29

TCF Technical Complexity Factor. 1, 27, 28

UAW Unadjusted Actor Weight. 1, 25

UCM Use Case Model. 1, 29

UCP Use Case Points. 1, 25, 28, 29

UPreW Unadjusted Precondition Weight. 1

UUCP Unadjusted Use Case Points. 1, 25, 26

UUCW Unadjusted Use Case Weight. 1, 25, 26

WEi Environmental Complexity Factor Weight. 1, 28

WMC Weighted Methods per Class. 1, 43

WTi Technical Complexity Factor Weight. 1, 27

References

Kelkar, K. (2017). *10 essential usability metrics.* Retrieved 2017-02-22, from https://www.userzoom.com/user-experience-research/top-ux-measurements-key-performance-indicators-usability-metrics

Mifsud, J. (2015). *Usability metrics – a guide to quantify the usability of any system.* Retrieved 2015-06-22, from https://usabilitygeek.com/usability-metrics-a-guide-to-quantify-system-usability

Sauro, J. (2011). *10 essential usability metrics.* Retrieved 2011-11-30, from https://measuringu.com/predicted-usability

Seeley, J. (2010). *Usability testing metrics.* Retrieved 2010-03-15, from https://designandresearch.wordpress.com/2010/03/15/usability-testing-metrics-jeff-seeley

West, J. (2015). *The hard truth about measuring page load time.* Retrieved 2015-06-23, from https://analyticsdemystified.com/google-analytics/hard-truth-measuring-page-load-time/