Team F

Deliverable 1

SOEN 6611 Software Measurement

Department of Software Engineering and Computer Science
Concordia University

DESCRIPTIVE-STATISTICS

Authors:
Amandeep Kaur Khosa
Dmitry Kryukov
Kritika Kritika
Mehak Jot Kaur
Roopamdeep Kaur
Sukhmeet Kaur

Instructor:
Prof. PANKAJ KAMTHAN

SUMMER 2018

Contents

0 Introduction

The purpose of the descriptive-statistics is engaged with the processing of empirical data, quantitative description of the set of information and data systematization.

The considered project is a module which can be integrated into the Online Grading System and help to improve users experience by using descriptive-statistics features. For this project the following functions were implemented from scratch: Minimum, Maximum, Mode, Median, Arithmetic Mean, and Standard Deviation. According to the requirements, no built-in functions were used. The new functions were written from scratch, and the whole project was written using the R language.

During the development cycle, the quality attributes such as efficiency, readability, maintainability, understandability, and usability were taken into account.

In the project GQM approach is used to achieve the goal of the project. [1] [2] [3] [4]

1 GQM

1.1 Goal

Create a module to **improve** the **efficiency** of studying process from the point of view of **students and professors** in the context of **Online Grading System**.

The descriptive-statistics in the module shall provide more efficient and accurate data based on new minimum, maximum, mode, median, mean and standard deviation functions for assigning the grades.

1.2 Questions

- **Q1**: How many lines of code on average do the files contain?
  M1: Number of files.
  M2: Number of lines of code (SLOC) in the files.
  M3: Average number of lines of code in the files.

- **Q2**: How to ensure that the descriptive-statistics functions return results in less than 2 seconds?
  M4: Time-based unit tests.

- **Q3**: How to ensure that all users satisfied with the module?
  M5: Student survey.

- **Q4**: What is the defect/failures density?
  M6: Number of failures reported by users per month.
  M2: Number of lines of code (SLOC) in the files.
  M7: Number of failures per KLOC.

- **Q5**: How to ensure that the code is maintainable?
  M8: Test coverage.
- **Q6**: How to ensure that the code is easy to read for other developers?
  M9: The code corresponds to the convention (code standard).
- **Q7**: How many functions are covered by the documentation?
  M10: Number of functions.
  M11: Number of comments.
  M12: The ratio of the number of comments to the SLOC.
- **Q8**: How many unit tests are performed?
  M13: The ratio of performed unit tests to the number of unit tests.
- **Q9**: How to ensure that the descriptive-statistics functions return the correct result?
  M8: Test coverage.
  M14: Number of unit tests.
- **Q10**: What are the bottlenecks of the increasing effectiveness of the system?
  M15: Action runtime.
- **Q11**: How to ensure that the module development is profitable?
  M16: Effort estimation.
  M17: Cost estimation.
  M18: Duration estimation.
- **Q12**: How effective are developers at finding failures?
  M14: Number of unit tests.
  M6: Number of failures reported by users per month.
  M13: The ratio of performed unit tests to the number of unit tests.

2 Use case model

Use case model represents the interaction between the actors and the system and shows the relationships between the actors and different use cases. [5]

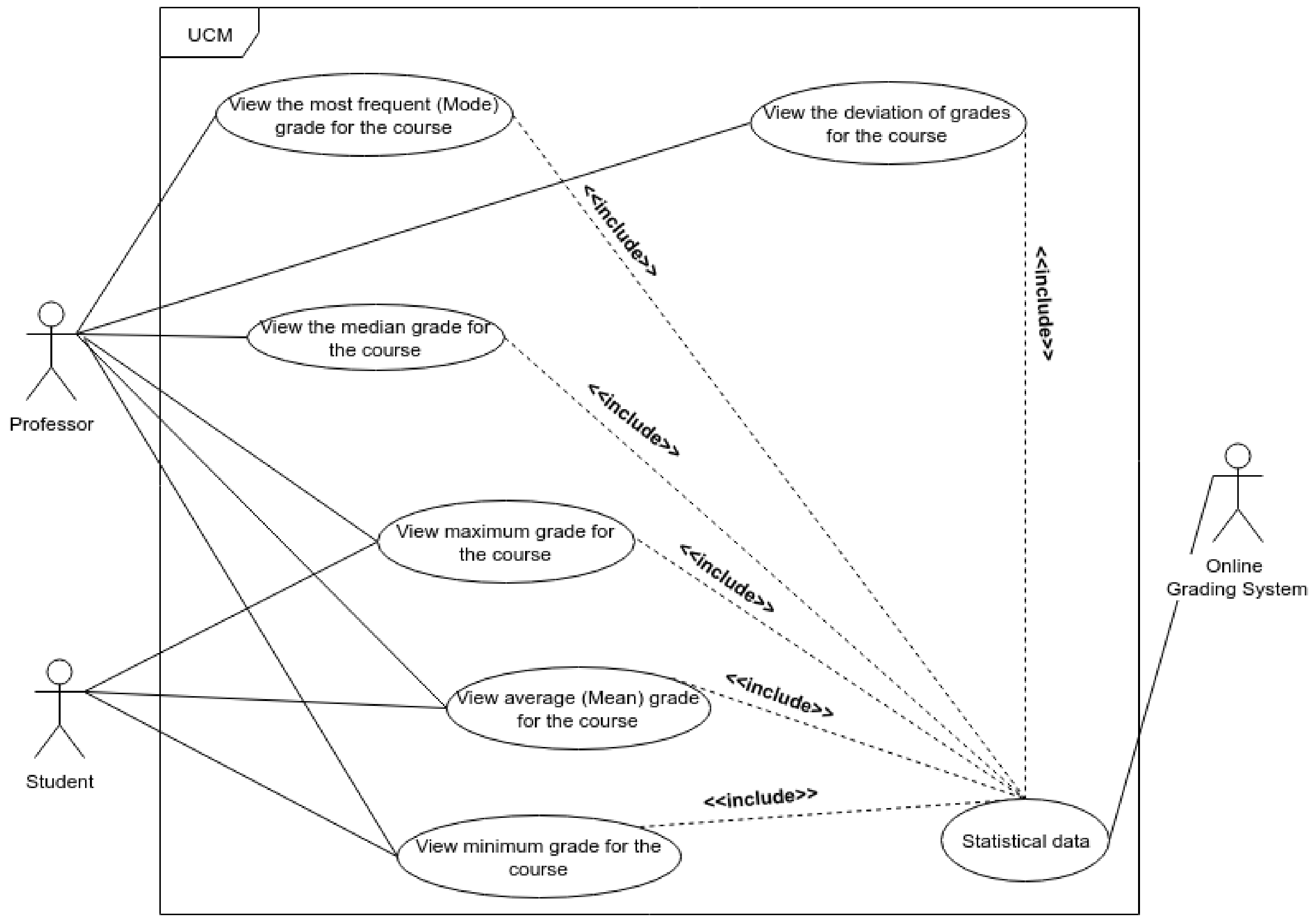

UCM
View the most frequent (Mode) grade for the course
View the deviation of grades for the course
View the median grade for the course
View maximum grade for the course
View average (Mean) grade for the course
View minimum grade for the course
Statistical data
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
Professor
Student
Online Grading System

Figure 1: Use case model

2.1 Use cases

Table 1: Use case - View the most frequent (Mode) grade for the course

Use case name	**View the most frequent (Mode) grade for the course**
Actors	Professor, Online Grading System (OGS).
Precondition	Professor has access to the system.
Postcondition	Professor can see the mode.
Main scenario (M)	1. Professor logins to the system. 2. The system shows the list of available courses. 3. Professor chooses the course to view. 4. The system shows the statistics of the course. 5. Professor views the mode of the course.
Extensions (E)	1.1. Professor entered the wrong credentials. 1.2. Go to 1.

Table 2: Use case - View the standard deviation for the course

Use case name	**View the standard deviation for the course**
Actors	Professor, Online Grading System (OGS).
Precondition	Professor has access to the system.
Postcondition	Professor views the deviation.
Main scenario (M)	1. Professor logins to the system. 2. The system shows the list of available courses. 3. Professor chooses the course to view. 4. The system shows the course statistics. 5. Professor views the standard deviation for the course.
Extensions (E)	1.1. Professor entered the wrong credentials. 1.2. Go to 1.

Table 3: Use case - View the median for the course

Use case name	**View the median for the course**
Actors	Professor, Online Grading System (OGS).
Precondition	Professor has access to the system.
Postcondition	Professor views the median.
Main scenario (M)	1. Professor logins to the system. 2. The system shows the list of available courses. 3. Professor chooses the course to view. 4. The system shows the course statistics. 5. Professor views the median.
Extensions (E)	1.1. Professor entered the wrong credentials. 1.2. Go to 1.

Table 4: Use case - View the average (Mean) grade for the course

Use case name	**View the average (Mean) grade for the course**
Actors	Professor, Student, Online Grading System (OGS).
Precondition	Professor has access to the system. Student has access to the system.
Postcondition	Professor views average (Mean). Student views average (Mean).
Main scenario (M)	1. Professor or Student logins to the system. 2. The system shows the list of available courses for the user (Professor or Student). 3. Professor or Student chooses the desirable course. 4. The system shows the average (Mean) grade for the course.
Extensions (E)	1.1. Professor or Student entered the wrong credentials 1.2. Go to 1

Table 5: Use case - View the minimum grade for the course

Use case name	**View the minimum grade for the course**
Actors	Professor, Student, Online Grading System (OGS).
Precondition	Professor has access to the system. Student has access to the system.
Postcondition	Professor views minimum. Student views minimum.
Main scenario (M)	1. Professor or Student logins to the system. 2. The system shows the list of available courses for the user (Professor or Student). 3. Professor or Student chooses the desirable course. 4. The system shows the minimum grade for the course.
Extensions (E)	1.1. Professor or Student entered the wrong credentials 1.2. Go to 1

Table 6: Use case - View the maximum grade for the course

Use case name	**View the maximum grade for the course**
Actors	Professor, Student, Online Grading System (OGS).
Precondition	Professor has access to the system. Student has access to the system.
Postcondition	Professor views maximum. Student views maximum .
Main scenario (M)	1. Professor or Student logins to the system. 2. The system shows the list of available courses for the user (Professor or Student). 3. Professor or Student chooses the desirable course. 4. The system shows the maximum grade for the course.
Extensions (E)	1.1. Professor or Student entered the wrong credentials. 1.2. Go to 1.

3 Estimates of effort

3.1 Use Case Points

The **UCP** is the effort estimation approach. To calculate the UCP the following values have to be calculated first: **UUCP** (Unadjusted Use Case Points), **TCF** (Technical Complexity Factor), **ECF** (Environment Complexity Factor). The formula to calculate the **UCP** is given below:

$$\mathbf{UCP} = \mathbf{UUCP} \times \mathbf{TCF} \times \mathbf{ECF} \tag{1}$$

The **UUCP** is calculated as the sum of **UAW** (Unadjusted Actor Weight) and **UUCW** (Unadjusted Use Case Weight):

$$\mathbf{UUCP} = \mathbf{UAW} + \mathbf{UUCW} \tag{2}$$

The **UAW** is calculated as the aggregated complexity of all the actors in all the use cases:

$$\mathbf{UAW} = \sum Actors * Weight \tag{3}$$

The classification of the actors in the use cases is shown in the table below:

Actor type	**Description**	**Weight**
A1	Simple actor	1
A2	Average actor	2
A3	Complex actor	3

Table 7: The classification of the actors and their associated weights in the UCP approach.

The system consists of three actors. Professor and student are complex actors, the descriptive-statistics module is a simple actor. Therefore:

$$\mathbf{UAW} = 1 \times 3 + 1 \times 3 + 1 \times 1 \tag{4}$$

$$\mathbf{UAW} = 7 \tag{5}$$

The **UUCW** is calculated as the total number of transactions in each use case.

$$\mathbf{UUCW} = \sum UseCase * Weight \tag{6}$$

Use case types and the corresponding weights are represented in the table below:

Use Case Type	**Description**	**Weight**
UC1	Simple Use Case	5
UC2	Average Use Case	10
UC3	Complex Use Case	15

Table 8: Use case types and the corresponding weights

Based on the **UCM** there are 6 simple use cases in the system. Therefore:

$$\mathbf{UUCW} = 6 \times 5 \tag{7}$$

$$\mathbf{UUCW} = 30 \tag{8}$$

In this case the **UUCP** is:

$$\mathbf{UUCP} = 7 + 30 \tag{9}$$

$$\mathbf{UUCP} = 37 \tag{10}$$

The purpose of **TCF** (Technical complexity factor) is to account the technical concerns that can impact the software project from its inception to its conclusion, including the delivery.

$$\mathbf{TCF} = C1 + \left[C2 \times \sum_{i=1}^{13} (W_{Ti} \times F_i) \right] \tag{11}$$

Where $C1 = 0.6$, $C2 = 0.01$, W_{Ti} is the Technical Complexity Factor Weight, and F_i is the Perceived Impact Factor corresponding to each Technical Complexity Factor.

TCF Type	**Description**	**Weight**	**Factor**
T1	Distributed System	2	0
T2	Performance	1	5
T3	End User Efficiency	1	4
T4	Complex Internal Processing	1	3
T5	Reusability	1	5
T6	Easy to Install	0.5	4
T7	Easy to Use	0.5	5
T8	Portability	2	5
T9	Easy to Change	1	5
T10	Concurrency	1	4
T11	Special Security Features	1	0
T12	Provides Direct Access for Third Parties	1	0
T13	Special User Training Facilities are Required	1	0

Table 9: The Technical Complexity Factors in the UCP approach.

In this case the **TCF** is:

$$\mathbf{TCF} = 0.6 + \left[0.01 \times (2\times0+1\times5+1\times4+1\times3+1\times5+0.5\times4+0.5\times5+2\times5+1\times5+1\times4+1\times0+1\times0+1\times0) \right] \tag{12}$$

$$\mathbf{TCF} = 1.005 \tag{13}$$

The purpose of **ECF** is to account the development team's personal traits, including the experience.

$$\mathbf{ECF} = C1 + \left[C2 \times \sum_{i=1}^{8} (W_{Ei} \times F_i) \right] \tag{14}$$

Where $C1 = 1.4$, $C2 = -0.03$, W_{Ei} is the Environmental Complexity Factor Weight, and F_i is the Perceived Impact Factor corresponding to each Environmental Complexity Factor.

ECF Type	Description	Weight	Factor
E1	Familiarity with the Use Case Domain	1.5	4
E2	Part-Time Workers	-1	3
E3	Analyst Capability	0.5	3
E4	Application Experience	0.5	4
E5	Object-Oriented Experience	1	5
E6	Motivation	1	5
E7	Difficult Programming Language	-1	5
E8	Stable Requirements	2	5

Table 10: The Environmental Complexity Factors in the UCP approach.

In this case the **ECF** is:

$$\mathbf{ECF} = 1.4 + \left[-0.03 \times (1.5 \times 4 + (-1 \times 3) + 0.5 \times 3 + 0.5 \times 4 + 1 \times 5 + 1 \times 5 + (-1 \times 5) + 2 \times 5) \right] \tag{15}$$

$$\mathbf{ECF} = 0.755 \tag{16}$$

Now the **UCP** can be calculated:

$$\mathbf{UCP} = 37 \times 1.005 \times 0.755 \tag{17}$$

$$\mathbf{UCP} = 28.075 \tag{18}$$

Since the team is new, the **PF** (Productivity Factor) is equal **20**. Therefore, the Effort Estimate is:

$$\textbf{Effort Estimate} = \mathbf{UCP} \times \mathbf{PF} \tag{19}$$

$$\textbf{Effort Estimate} = 28.075 \times 20 \tag{20}$$

$$\textbf{Effort Estimate} = 561.5 \quad \text{person-hours} \tag{21}$$

Therefore, the **estimated effort** of the project by using the **UCP** approach is **562** person-hours.

3.2 COCOMO 81

The COCOMO (Constructive Cost Model) approach is a cost model. Based on the empirical data it provides the effort estimation or duration of the project.

$$\textbf{Effort Estimation (E)} = a \times (S)^b \times F \quad \text{person-month} \tag{22}$$

Where **S** is software size in Kilo Delivered Source Instructions (KDSI), the coefficients **a** and **b** depend on the type of software project (as shown in Table 11), and **F** is the adjustment factor. In Basic COCOMO, there is no adjustment, so **F = 1**.

The equation for the duration estimation in Basic COCOMO is:

$$\textbf{Duration Estimation (D)} = c \times (E)^d \quad \text{time in month} \tag{23}$$

Where coefficients **c** and **d** depend on the type of the project (see Table 11).

The equation for the cost estimation in Basic COCOMO is:

$$\textbf{Cost Estimation (P)} = E/D \quad \text{people required} \tag{24}$$

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Table 11: Types of the project.

Based on the implemented module the project is Organic with a small team. So the **S** = 0.236 KDSI, **F = 1** and the coefficients **a** = 2.4, **b** = 1.05, **c** = 2.5,**d** = 0.38. Therefore:

$$\textbf{Effort Estimation (E)} = 2.4 \times (0.236)^{1.05} \times 1 \quad \text{person-month} \tag{25}$$

$$\mathbf{E} = 0.527 \quad \text{person-month} \tag{26}$$

$$\textbf{Duration Estimation (D)} = 2.5 \times (0.527)^{0.38} \quad \text{time in month} \tag{27}$$

$$\mathbf{D} = 1.96 \quad \text{time in month} \tag{28}$$

$$\textbf{Cost Estimation (P)} = 0.527/1.96 \quad \text{people required} \tag{29}$$

$$\mathbf{P} = 0.269 \quad \text{people required} \tag{30}$$

3.3 Difference in the estimates using UCP approach and COCOMO

The effort estimation by **UCP (E) = 562** person-hours, by **COCOMO (E) = 0.527** person-month or **COCOMO (E) = 92.7** person-hours by taking a working hours per month as 176 hours (22 days, 8 hours per day).

Based on the actual time spent by the team on the project, the **COCOMO** estimation is more **accurate** and close to reality. The difference in the estimation by using **UCP** approach and **COCOMO** is more than 469 person-hours.

4 Implementation

4.1 Code listings

In this patt the code listings of the implemented functions are represented. The comments are excluded from the listings to save the space.

4.1.1 Init file

The file that calls all the implemented functions of the descriptive-statistics.

```r
source("stdlib.R")
source("min.R")
source("max.R")
source("mode.R")
source("median.R")
source("mean.R")
source("deviation.R")

array <- sample(0:100, 100, replace=TRUE)
print(paste0("Min: ", SoenMin(array)))
print(paste0("Max: ", SoenMax(array)))
print(paste0("Mode: ", SoenMode(array)))
print(paste0("Median: ", SoenMedian(array)))
print(paste0("Mean: ", SoenMean(array)))
print(paste0("Deviation: ", SoenDeviation(array)))
```

4.1.2 Min function

In the context of the Online Grading System this function returns the lowest grade of the course. This value doesn't need to be unique. This function finds the minimum value in the given array and returns it.

```r
source("stdlib.R")

SoenMin <- function(array){
  l <- len(array)
  if (l == 0) {
    return("Empty array")
  }
  if (l == 1) {
    return(array[l])
  }
  m <- 9999999999999999
  for (elem in array) {
    if (elem < m) {
      m <- elem
    }
  }
  return(m)
}
```

4.1.3 Max function

In the context of the Online Grading System this function returns the highest grade of the course. This value doesn't need to be unique. This function finds the maximum value in the given array and returns it.

```r
source("stdlib.R")

SoenMax <- function(array){
  l <- len(array)
  if (l == 0) {
    return("Empty array")
  }
  if (l == 1) {
    return(array[1])
  }
 M <- -9999999999999999
  for (elem in array) {
    if (elem > M) {
      M <- elem
    }
  }
  return(M)
}
```

4.1.4 Mode function

This function finds the most frequent value or values in the given array. In the context of the Online Grading System this function finds the most frequent grade (or the set of grades) for the course. Mode does not need to be unique. There can be more than one mode. This function works with the sorted array.

```r
source("stdlib.R")

SoenMode <- function(array){
  l <- len(array)
  if (l == 0) {
    return("Empty array")
  }
  if (l == 1) {
    return(array[1])
  }
  array <- SoenSort(array)
  max.counter <- 1
  current.counter <- 1
  o <- vector()
  for (elem in c(2:l)) {
    if (array[elem] == array[elem - 1]) {
      current.counter <- current.counter + 1
    } else {
        if (current.counter == max.counter) {
          o <- c(o, array[elem-1])
        }
        if (current.counter > max.counter) {
          max.counter <- current.counter
          o <- array[elem-1]
        }
        current.counter <- 1
      }
  }

  if (current.counter > max.counter) {
    o <- array[l]
  }
  if (max.counter == current.counter) {
    o <- c(o, array[l])
  }
  return(o)
}
```

4.1.5 Median function

This function returns the middle number of an odd array or the arithmetic mean of the two middle numbers of an even array. This function works with the sorted array and returns the middle grade of the course.

```
source("stdlib.R")
source("mean.R")

SoenMedian <- function(array){
  l <- len(array)
  if(l == 0){
    return("Empty array")
  }
  if(l == 1){
    return(array[l])
  }
  array <- SoenSort(array)
  if (l%%2 == 0){
    d <- SoenMean(c(array[l/2],array[l/2+1]))
  } else {
    d <- array[l/2+0.5]
  }

  return(d)
}
```

4.1.6 Arithmetic Mean function

This function returns the average value of the given array. In the context of the Online Grading System it calculates the average grade of the course.

```
source("stdlib.R")

SoenMean <- function(array){
  l <- len(array)
  mu <- 0
  if (l == 0) {
    return("Empty array")
  }
  if (l == 1) {
    return(array[l])
  }
  for (elem in array) {
    mu <- mu+elem
  }
  mu <- mu/l
  return(mu)
}
```

4.1.7 Standard Deviation function

This function calculates the standard deviation value for a total population of the given array.

```
source("stdlib.R")
source("mean.R")

SoenDeviation <- function(array){
  l <- len(array)
  if (l == 0) {
    return("Empty array")
  }
  if (l == 1) {
    return("Now enough data")
  }
  mu <- SoenMean(array)
  omega <- 0

  for (elem in array) {
    omega <- omega+(elem-mu)^2
  }
  omega <- omega/l
  omega <- SoenSqrtA(omega)
  return(omega)
}
```

4.1.8 Library with standard functions

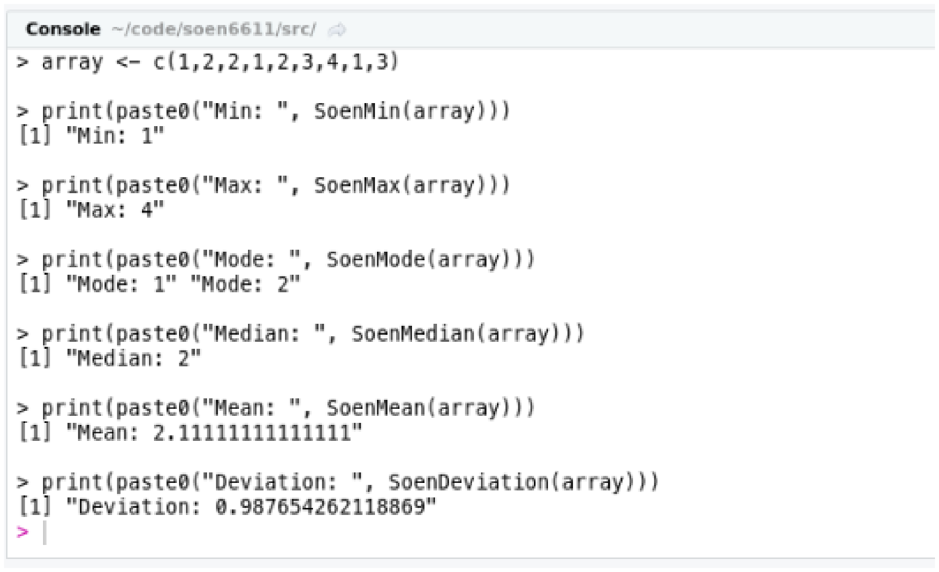
The class with implemented standard mathematical functions is represented below. It includes the calculation of the array length, square root and the quick sort. All these methods were used in descriptive-statistics functions.

```
# Return the length of an array
SoenLen <- function(array){
  count <- 0
  for (elem in array){
    count <- count+1
  }
  return(count)
}

# Return sqrt of a number
SoenSqrtA <- function(number){
  if (number == 0) {
    return(number)
  }
```

```r
  if (number == 1) {
    return(number)
  }
  low = 0
  high = number
  mid = 0

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

# Return sqrt of a number
SoenSqrtB <- function(number){
  i <- 0
  j <- number/2+1

  while (i <= j){
    mid <- (i+j)/2
    if (number/mid == mid) {
      return(mid)
    } else if (mid < number/mid) {
      i <- mid +1
    } else {
      j <- mid -1
    }
  }
  return(j)
}

# Return the sorted array via quick sort
SoenSort <- function(array){
  l <- len(array)
  if (l > 1) {
    p = array[l %/% 2]
    left = sortB(array[array < p])
    middle = array[array == p]
    right =  sortB(array[array > p])
    return(c(left, middle, right))
  } else {
    return(array)
  }
}
```

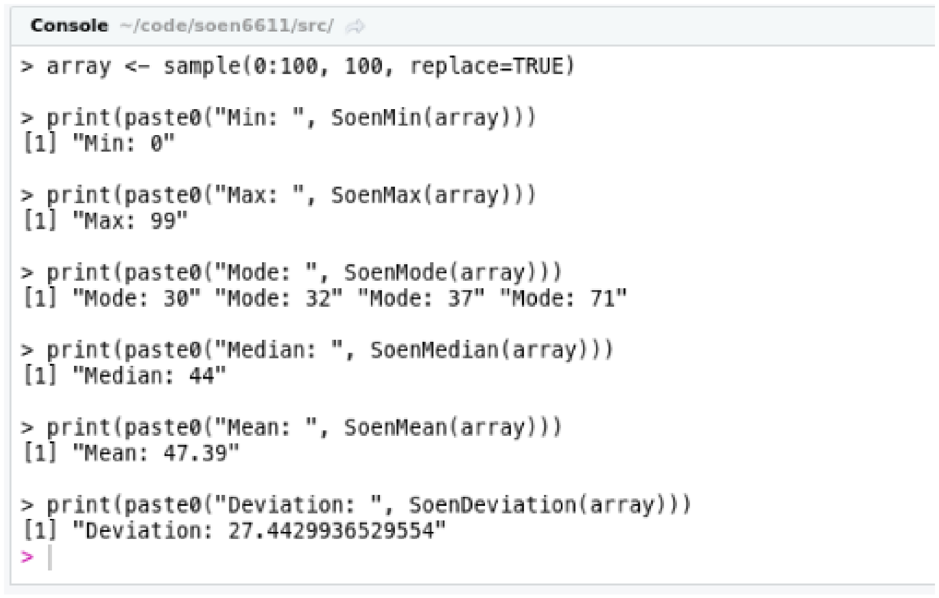
4.2 Testing

In this section the code for testing and testing results are represented. It's necessary to install package **testthat** before the start of the unit tests.



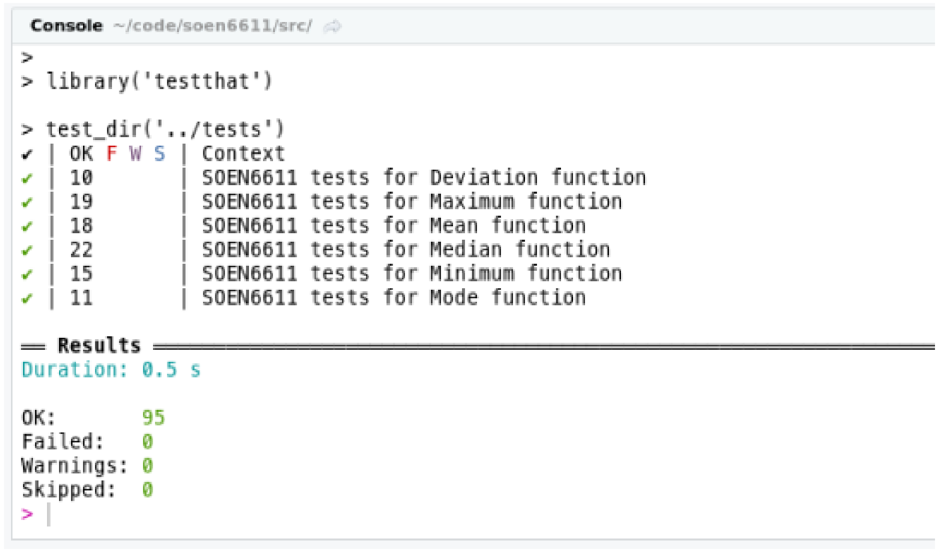
```
Console ~/code/soen6611/src/
> array <- c(1,2,2,1,2,3,4,1,3)

> print(paste0("Min: ", SoenMin(array)))
[1] "Min: 1"

> print(paste0("Max: ", SoenMax(array)))
[1] "Max: 4"

> print(paste0("Mode: ", SoenMode(array)))
[1] "Mode: 1" "Mode: 2"

> print(paste0("Median: ", SoenMedian(array)))
[1] "Median: 2"

> print(paste0("Mean: ", SoenMean(array)))
[1] "Mean: 2.11111111111111"

> print(paste0("Deviation: ", SoenDeviation(array)))
[1] "Deviation: 0.987654262118869"
> 
```

Figure 2: Results with the specified array



```
Console ~/code/soen6611/src/
> array <- sample(0:100, 100, replace=TRUE)

> print(paste0("Min: ", SoenMin(array)))
[1] "Min: 0"

> print(paste0("Max: ", SoenMax(array)))
[1] "Max: 99"

> print(paste0("Mode: ", SoenMode(array)))
[1] "Mode: 30" "Mode: 32" "Mode: 37" "Mode: 71"

> print(paste0("Median: ", SoenMedian(array)))
[1] "Median: 44"

> print(paste0("Mean: ", SoenMean(array)))
[1] "Mean: 47.39"

> print(paste0("Deviation: ", SoenDeviation(array)))
[1] "Deviation: 27.4429936529554"
> 
```

Figure 3: Results with a random array



```
Console ~/code/soen6611/src/
>
> library('testthat')

> test_dir('../tests')
✔ | OK F W S | Context
✔ | 10       | SOEN6611 tests for Deviation function
✔ | 19       | SOEN6611 tests for Maximum function
✔ | 18       | SOEN6611 tests for Mean function
✔ | 22       | SOEN6611 tests for Median function
✔ | 15       | SOEN6611 tests for Minimum function
✔ | 11       | SOEN6611 tests for Mode function

══ Results ═══════════════════════════════════════════════
Duration: 0.5 s

OK:       95
Failed:   0
Warnings: 0
Skipped:  0
>
```

Figure 4: Tests report

4.2.1 Test initiator

The main test file for starting all the tests

```
library('testthat')

test_dir('../tests')
```

4.2.2 Tests for the Minimum function

The set of tests for the minimum function:

```
context("SOEN6611 tests for the Minimum function")

test_that("Positive arrays", {
  expect_that(SoenMin(c(1,2,3,4,5,6)), equals(1))
  expect_that(SoenMin(c(1,2,3,1,1,1)), equals(1))
  expect_that(SoenMin(c(1,1)), equals(1))
  expect_that(SoenMin(c(1)), equals(1))
  expect_that(SoenMin(c(4)), equals(4))
  expect_that(SoenMin(c(5,8,6,7,5,6,9,8,6,5,7,4,7,8,9,6)), equals(4))
  expect_that(SoenMin(c(4,2,3,4,5,1)), equals(1))
})

test_that("Negative arrays", {
  expect_that(SoenMin(c(1,-2,3,-4,5,6)), equals(-4))
  expect_that(SoenMin(c(-1,-2,-3,-1,-1,-1)), equals(-3))
  expect_that(SoenMin(c(-1,-1)), equals(-1))
  expect_that(SoenMin(c(-1)), equals(-1))
  expect_that(SoenMin(c(-6)), equals(-6))
  expect_that(SoenMin(c(-5,8,-6,7,5,-6,9,8,-6,5,7,-4,7,8,9,6)), equals(-6))
  expect_that(SoenMin(c(4,-2,-3,4,5,-1)), equals(-3))
})

test_that("Big arrays", {
  arr <- sample(0:100, 100, replace=TRUE)
  expect_that(SoenMin(arr), equals(min(arr)))
})
```

4.2.3 Tests for the Maximum function

The set of tests for the maximum function.

```
context("SOEN6611 tests for Maximum function")

test_that("Positive arrays", {
  expect_that(SoenMax(c(1,2,3,4,5,6)), equals(6))
  expect_that(SoenMax(c(1,2,3,1,1,1)), equals(3))
  expect_that(SoenMax(c(5,5,5,5,5,5,5,5,5)), equals(5))
  expect_that(SoenMax(c(1,1)), equals(1))
  expect_that(SoenMax(c(1)), equals(1))
  expect_that(SoenMax(c(8)), equals(8))
  expect_that(SoenMax(c(5,8,6,7,5,6,9,8,6,5,7,4,7,8,9,6)), equals(9))
  expect_that(SoenMax(c(4,2,3,4,5,5)), equals(5))
  expect_that(SoenMax(c(4,2,3,4,3,5)), equals(5))
})

test_that("Negative arrays", {
  expect_that(SoenMax(c(1,-2,3,-4,5,-6)), equals(5))
  expect_that(SoenMax(c(-1,-2,-3,-1,1,-1)), equals(1))
  expect_that(SoenMax(c(-5,-5,-5,-5,-5,-5,-5,-5,-5)), equals(-5))
  expect_that(SoenMax(c(1,-1)), equals(1))
  expect_that(SoenMax(c(-1)), equals(-1))
  expect_that(SoenMax(c(-8)), equals(-8))
  expect_that(SoenMax(c(5,-8,6,-7,5,6,-9,-8,6,5,-7,4,7,-8,-9,6,99)), equals(99))
  expect_that(SoenMax(c(-4,-2,-3,-4,-5,-5)), equals(-2))
  expect_that(SoenMax(c(4,-2,3,-4,3,5)), equals(5))
})

test_that("Big arrays", {
  arr <- sample(0:100, 100, replace=TRUE)
  expect_that(SoenMax(arr), equals(max(arr)))
})
```

4.2.4 Tests for the Mode function.

The set of tests for the mode function.

```
context("SOEN6611 tests for Mode function")

test_that("Positive arrays", {
  expect_that(SoenMode(c(1,3,4,2,5,4,5)), equals(c(4,5)))
  expect_that(SoenMode(c(1,1,1,2,2,3,4,5,2)), equals(c(1,2)))
  expect_that(SoenMode(c(1,1,1,2,2,3,4,5,2,1)), equals(1))
  expect_that(SoenMode(c(1)), equals(1))
  expect_that(SoenMode(c(5)), equals(5))
})

test_that("Negative arrays", {
  expect_that(SoenMode(c(1,3,-4,2,5,-4,5)), equals(c(-4,5)))
  expect_that(SoenMode(c(-1,1,1,-2,2,3,4,5,2)), equals(c(1,2)))
  expect_that(SoenMode(c(-1,-1,-1,2,2,3,4,5,2,-1)), equals(-1))
  expect_that(SoenMode(c(-1)), equals(-1))
  expect_that(SoenMode(c(-4)), equals(-4))
})

test_that("Big arrays", {
  arr <- sample(0:100, 100, replace=TRUE)
  expect_that(SoenMean(arr), is_a('numeric'))
})
```

4.2.5 Tests for the Median function.

The set of tests for the median function.

```
context("SOEN6611 tests for Median function")

test_that("Positive arrays even", {
  a1 <- c(1,2,3,2,4,3,2,4,3,2)
  a2 <- c(1,1)
  a3 <- c(9,9)
  a4 <- c(3,2,1,4,3,2,1,3)
  a5 <- c(1,1,1,1,1,1,1,1)
  expect_that(SoenMedian(a1), equals(median(a1)))
  expect_that(SoenMedian(a2), equals(median(a2)))
  expect_that(SoenMedian(a3), equals(median(a3)))
  expect_that(SoenMedian(a4), equals(median(a4)))
  expect_that(SoenMedian(a5), equals(median(a5)))
})

test_that("Positive arrays odd", {
  b1 <- c(1,3,2,6,5,8,3)
  b2 <- c(1)
  b3 <- c(0)
  b4 <- c(1,1,1)
  b5 <- c(1,5,4,5,4,1,1)
  expect_that(SoenMedian(b1), equals(median(b1)))
  expect_that(SoenMedian(b2), equals(median(b2)))
  expect_that(SoenMedian(b3), equals(median(b3)))
  expect_that(SoenMedian(b4), equals(median(b4)))
  expect_that(SoenMedian(b5), equals(median(b5)))
})

test_that("Negative arrays even", {
  c1 <- c(-1,-4,-2,-5,-4,-3)
  c2 <- c(-1,-1)
  c3 <- c(-9,9)
  c4 <- c(-9,-9)
  c5 <- c(-1,-3,2,-2)
  expect_that(SoenMedian(c1), equals(median(c1)))
  expect_that(SoenMedian(c2), equals(median(c2)))
  expect_that(SoenMedian(c3), equals(median(c3)))
  expect_that(SoenMedian(c4), equals(median(c4)))
  expect_that(SoenMedian(c5), equals(median(c5)))

})

```

```
test_that("Negative arrays odd", {
  d1 <- c(-1,-4,-3,-5,-6,-32,-7)
  d2 <- c(-1,5,6,3,-4,5,-6)
  d3 <- c(-1)
  d4 <- c(-1,-1,-1)
  d5 <- c(-1,1,-1)
  expect_that(SoenMedian(d1), equals(median(d1)))
  expect_that(SoenMedian(d2), equals(median(d2)))
  expect_that(SoenMedian(d3), equals(median(d3)))
  expect_that(SoenMedian(d4), equals(median(d4)))
  expect_that(SoenMedian(d5), equals(median(d5)))

})

test_that("Big arrays", {
  arr <- sample(0:100, 100, replace=TRUE)
  expect_that(SoenMedian(arr), equals(median(arr)))

  arr2 <- sample(0:4000, 1000, replace=TRUE)
  expect_that(SoenMedian(arr2), equals(median(arr2)))
})
```

4.2.6 Tests for the Mean function.

The set of tests for the arithmetic mean function.

```
context("SOEN6611 tests for Mean function")

test_that("Positive arrays", {
  expect_that(SoenMean(c(1,2,3,4,5,6)), equals(3.5))
  expect_that(SoenMean(c(1,2,3,1,1,1)), equals(1.5))
  expect_that(SoenMean(c(1,1)), equals(1))
  expect_that(SoenMean(c(1)), equals(1))
  expect_that(SoenMean(c(6)), equals(6))
  expect_that(SoenMean(c(5,8,6,7,5,6,9,8,6,5,7,4,7,8,9,6)), equals(6.625))

  a1 <- c(3,5,3,6,7,6,5,3,2,4,3,4,32)
  a2 <- c(0,0,0,0)
  a3 <- c(0)
  a4 <- c(1,0,0,0,0)
  a5 <- c(1,3)
  expect_that(SoenMean(a1), equals(mean(a1)))
  expect_that(SoenMean(a2), equals(mean(a2)))
  expect_that(SoenMean(a3), equals(mean(a3)))
  expect_that(SoenMean(a4), equals(mean(a4)))
  expect_that(SoenMean(a5), equals(mean(a5)))
})

test_that("Negative arrays", {
  a1 <- c(-3,-5,-3,6,-7,6,5,3,-2,4,3,-4,32)
  a2 <- c(-0,-0,-0,-0)
  a3 <- c(-0)
  a4 <- c(-1,-0,-0,-0,-0)
  a5 <- c(1,-3)
  a6 <- c(-3,-5,-3,-7,-2,-4,-5,-3)
  expect_that(SoenMean(a1), equals(mean(a1)))
  expect_that(SoenMean(a2), equals(mean(a2)))
  expect_that(SoenMean(a3), equals(mean(a3)))
  expect_that(SoenMean(a4), equals(mean(a4)))
  expect_that(SoenMean(a5), equals(mean(a5)))
  expect_that(SoenMean(a6), equals(mean(a6)))
})

test_that("Big arrays", {
  arr <- sample(0:100, 100, replace=TRUE)
  expect_that(SoenMean(arr), equals(mean(arr)))
})
```

4.2.7 Tests for the Standard Deviation function.

The set of tests for the standard deviation function.

```
context("SOEN6611 tests for Deviation function")

test_that("Positive arrays", {
  a1 <- c(1,4,5,3,1,5,4)
  a2 <- c(1,1)
  a3 <- c(13,5,2,3,21,3,2)
  a4 <- c(1,1,1,1,1)
  expect_that(SoenDeviation(a1), equals(sd(a1)*(sqrt((length(a1)-1)/length(a1)))))
  expect_that(SoenDeviation(a2), equals(sd(a2)*(sqrt((length(a2)-1)/length(a2)))))
  expect_that(SoenDeviation(a3), equals(sd(a3)*(sqrt((length(a3)-1)/length(a3)))))
  expect_that(SoenDeviation(a4), equals(sd(a4)*(sqrt((length(a4)-1)/length(a4)))))
})

test_that("Negative arrays", {
  c1 <- c(1,-4,3,-2,-4)
  c2 <- c(-1,-1)
  c3 <- c(-1,-1,-1,-1)
  c4 <- c(-1,3,-12,4,-1)
  expect_that(SoenDeviation(c1), equals(sd(c1)*(sqrt((length(c1)-1)/length(c1)))))
  expect_that(SoenDeviation(c2), equals(sd(c2)*(sqrt((length(c2)-1)/length(c2)))))
  expect_that(SoenDeviation(c3), equals(sd(c3)*(sqrt((length(c3)-1)/length(c3)))))
  expect_that(SoenDeviation(c4), equals(sd(c4)*(sqrt((length(c4)-1)/length(c4)))))

})

test_that("Big arrays", {
  arr <- sample(0:100, 100, replace=TRUE)
  expect_that(SoenDeviation(arr), equals(sd(arr)*(sqrt((length(arr)-1)/length(arr))
    )))

  arr2 <- sample(0:4000, 1000, replace=TRUE)
  expect_that(SoenDeviation(arr2), equals(sd(arr2)*(sqrt((length(arr2)-1)/length(
    arr2)))))
})
```

5 Cyclomatic complexity

Cyclomatic complexity is a software metric used to measure the complexity of a program. This metric measures independent paths through program source code. Cyclomatic complexity can be calculated with respect to functions, modules, methods or classes within a program.

$$\textbf{Complexity of the Program V(G)} = E - N + 2 \tag{31}$$

Where, E - Number of edges, N - Number of Nodes in the Flow graph.

$$\textbf{Complexity of the Program V(G)} = P + 1 \tag{32}$$

Where P = Number of node that contains condition. [6]

Cyclomatic number for each function of the module is represented in the table below:

File	**Function**	**Cyclomatic number**
init.R	Flow of calls	1
min.R	SoenMin()	5
max.R	SoenMax()	5
mode.R	SoenMode()	9
median.R	SoenMedian()	4
mean.R	SoenMean()	4
deviation.R	SoenDeviation()	4
stdlib.R	SoenLen()	2
	SoenSqrtA()	5
	SoenSqrtB()	4
	SoenSort()	2
Total		**45**

Table 12: Cyclomatic complexity for each function.

The overall **Cyclomatic Number** for the descriptive-statistics project is **45**.

Conclusion: According to the Range-Risk distribution of complexity number, each class in the project has a **simple** control flow, less than 10. But the whole project has cyclomatic number equal 45, that falls under category of High risk assessment. The source code has a very complex, but manageable, control flow.

6 Object-oriented metrics

In this section several metrics are calculated: **WMC** (Weighted Method per Class), **CF** (Coupling Factor), **LCOM** (Lack of Cohesion in Methods). The class diagram for the project is represented below. Based on this diagram, **WMC**, **CF** and **LCOM** can be calculated.

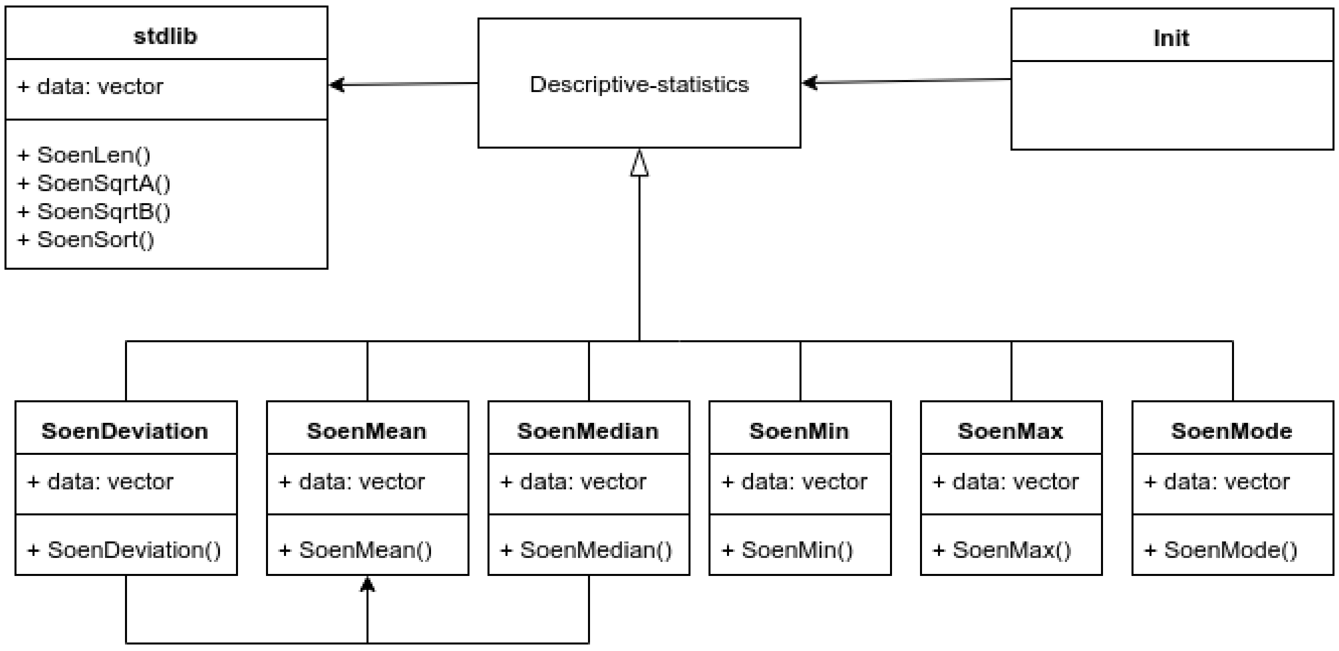


Figure 5: Class diagram

6.1 Weighted Methods per Class

The weights for the methods per class are represented in the table below.

Class	Method	Weight
init.R	Flow of calls	1
min.R	SoenMin()	5
max.R	SoenMax()	5
mode.R	SoenMode()	9
median.R	SoenMedian()	4
mean.R	SoenMean()	4
deviation.R	SoenDeviation()	4
stdlib.R	SoenLen()	2
	SoenSqrtA()	5
	SoenSqrtB()	4
	SoenSort()	2

Table 13: WMC.

WMC is calculated by using the following formula:

$$\textbf{WMC} = \sum_{i=1}^{n} c_i(M_i) \tag{33}$$

Where C a class with methods M1 to Mn. Let c1(M1), to cn(Mn) be the complexity (weights) of the methods [7]. Therefore:

$$\textbf{WMC} = 1 + 5 + 5 + 9 + 4 + 4 + 4 + 2 + 5 + 4 + 2 = 45 \tag{34}$$

6.2 Coupling Factor

The coupling factor (CF) measures the average coupling between classes. **CF** is calculated using the following formula:

$$\textbf{CF} = \frac{\sum_{i=1}^{n} \left[\sum_{j=1}^{n} isClient(C_i, C_j) \right]}{n^2 - n} \tag{35}$$

isClient(C_i, C_j) in the formula means that the class C_i has a relationship with the class C_j that is not inheritance. Based on the class diagram, the coupling factor is:

$$\textbf{CF} = \frac{0 + 6 + 2 + 1 + 2 + 1 + 1 + 1}{8^2 - 8} \tag{36}$$

$$\textbf{CF} = 0.25 \tag{37}$$

The **Coupling Factor** for this project is equal to 0.25.

6.3 Lack of Cohesion in Methods

The Lack of Cohesion in methods (**LCOM**) is calculated by using the following formula:

$$LCOM = \frac{\frac{1}{a}\left[\sum_{i=1}^{a}\mu(A_i)\right] - m}{1-m} \tag{38}$$

For each attribute we have to know the number of methods that access this attribute. **LCOM** for each of the classes is calculated below:

$$LCOM_{stdlib} = \frac{\frac{1}{2}\times[2+2]-4}{1-4} = \frac{2}{3} \tag{39}$$

$$LCOM_{min} = \frac{\frac{1}{1}\times 1-1}{1-1} = \frac{0}{0} = 0 \tag{40}$$

$$LCOM_{max} = \frac{\frac{1}{1}\times 1-1}{1-1} = \frac{0}{0} = 0 \tag{41}$$

$$LCOM_{mode} = \frac{\frac{1}{1}\times 1-1}{1-1} = \frac{0}{0} = 0 \tag{42}$$

$$LCOM_{median} = \frac{\frac{1}{1}\times 1-1}{1-1} = \frac{0}{0} = 0 \tag{43}$$

$$LCOM_{mean} = \frac{\frac{1}{1}\times 1-1}{1-1} = \frac{0}{0} = 0 \tag{44}$$

$$LCOM_{deviation} = \frac{\frac{1}{1}\times 1-1}{1-1} = \frac{0}{0} = 0 \tag{45}$$

For six classes **LCOM** is 0 and that means that these classes have maximum cohesion. Class stdlib has **LCOM** $= \frac{2}{3}$ and that means that cohesion in this class is not high, but not too low.

7 SLOC

The most common definition of the **Physical SLOC** is a count of lines of the source code excluding comment lines. So, manually our **Physical SLOC = 265**. The **Logical SLOC** is a number of the executable statements, excluding the comments. The **Logical SLOC** is language dependable. Manually our **Logical SLOC = 234**.

Physycal SLOC = 265 and **Logical SLOC = 236** according to the GitHub code editor used as a tool for calculating the SLOC.

The manual calculation of the **Physical SLOC** and the use of the tool showed the same result, so the total **Physical SLOC** is equal to 265.

The manual calculation of the **Logical SLOC** and the use of the tool showed different results (234 and 236 respectively). The difference probably appeared because the tool considers the executable statements in a different way that we did during the manual calculation.

8 Correlations

8.1 Scatter plot

WMC and the **Logical SLOC** for each class are represented in the table below:

Class	**WMC**	**Logical SLOC**
init	1	25
min	5	21
max	5	21
mode	9	39
median	4	23
mean	4	20
deviation	4	23
stdlib	13	64

Table 14: WMC and LSLOC for each class.

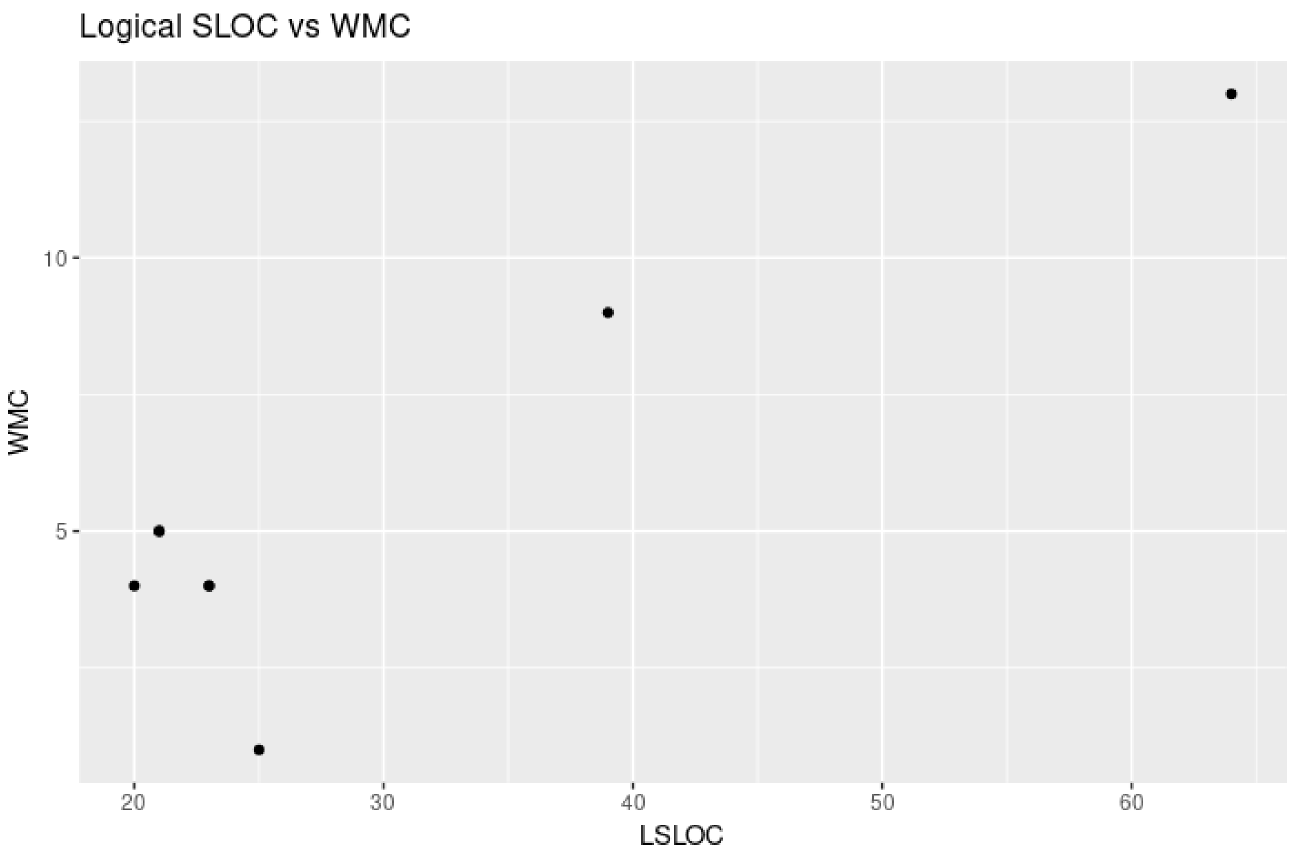


Figure 6: Scatter Plot

Based on the represented Scatter Plot, there is a correlation between the **Logical SLOC** and **WMC**. If the **Logical SLOC** increases, **WMC** increases as well.

8.2 Correlation coefficient

Since the attributes are not distributed normally, we can apply the **Spearman's Rank Correlation Coefficient** r_s. [8]

$$r_s = 1 - \frac{6 \times \sum_{i=1}^{n} d_i^2}{n^3 - n} \tag{46}$$

Class	**WMC**(x_i)	**Rank**(x_i)	**Logical SLOC**(y_i)	**Rank**(y_i)	**d**	d^2
init	1	1	25	6	-5	25
min	5	5.5	21	2.5	3.5	12.25
max	5	5.5	21	2.5	3.5	12.25
mode	9	7	39	7	0	0
median	4	3	23	4.5	-1.5	2.25
mean	4	3	20	1	2	4
deviation	4	3	23	4.5	-1.5	2.25
stdlib	13	8	64	8	0	0

Table 15: Data for Spearman's correlation coefficient

Therefore:

$$r_s = 1 - \frac{6 \times (25 + 12.25 + 12.25 + 0 + 2.25 + 4 + 2.25 + 0)}{8^3 - 8} \tag{47}$$

$$r_s = 1 - \frac{348}{504} \tag{48}$$

$$r_s = 0.31 \tag{49}$$

Based on the **Spearman's Rank Correlation Coefficient**, we have a positive correlation between **WMC** and the **Logical SLOC** for all the classes.

9 Glossary

Term	**Definition**
GQM	Goal Question Metric
UCM	Use Case Model
UCP	Use Case Points
UUCP	Unadjusted Use Case Points
UUCW	Unadjusted Use Case Weight
UAW	Unadjusted Actor Weight
TCF	Technical Complexity Factor
ECF	Environment Complexity Factor
PF	Productivity Factor
COCOMO	Constructive Cost Model
Cyclomatic number	Indicator of internal complexity
WMC	Weighted Method per Classes
CF	Coupling Factor
LCOM	Lack of Cohesion in Methods
SLOC	Source Line Of Code
OGS	Online Grading System
r_s	Spearman Rank Correlation Coefficient

Table 16: Glossary

References

[1] John Tanner. Agile metrics: A gqm approach. 2017. https://www.leadingagile.com/2017/05/agile-metrics-gqm-approach/.

[2] Christopher M. Lott Christiane Differding, Barbara Hoisl. *Technology Package for the Goal Question Metric Paradigm.* 1996.

[3] Raf Cammarano. Goal question metric (gqm) model. 2008. https://rafcammarano.wordpress.com/2008/04/07/goal-question-metric-gqm-model/.

[4] Bjorn Lindstrom. *A Software Measurement Case Study using GQM.*

[5] Wikipedia. Use case diagram. 2018. https://en.wikipedia.org/wiki/Use_case_diagram.

[6] Guru99. Learn mccabe's cyclomatic complexity with example. unknown year. https://www.guru99.com/cyclomatic-complexity.html.

[7] Pankaj Kamthan. Metrics for classes in object-oriented design. 2018. http://users.encs.concordia.ca/~kamthan/courses/soen-6611/ood_metrics_classes.pdf.

[8] Pankaj Kamthan. Introduction to analysis of software measurement data. 2018. http://users.encs.concordia.ca/~kamthan/courses/soen-6611/software_measurement_data_analysis.pdf.