



EVOLUTION OF THE MAINTAINABILITY IN OPEN SOURCE OFFICE COLLABORATION TOOLS SOEN-6611

Team 4-Sidharth Sharma

Dhruv Gusain

Uhegbu Emmanuel

Uchechukwu Iroegbu

Deepika Sharma

Guneet Saini

Faculty Advisor-Dr. Rodrigo Morales

Department of Computer Science And Software Engineering

Concordia University

OUTLINE

- Introduction
- Background/Literature Review
- Research questions
- Research methods
- Data Collection
- Discussion
- Conclusion
- Future Research

INTRODUCTION

- In our project we analyzed distinct versions of **Apache openoffice** and **libreoffice** and compared the evolution of software maintainability by calculating and comparing the related metrics of each release.
- Volume, Unit size, Complexity and code duplication are the four internal characteristics, that were taken into consideration to reach the external attributes mainly, Analyzability, Changeability, Testability.
- Maintainability model is then generated to have a broader inspection of maintainability, in terms of dependency rather than generating a quantifying value.

BACKGROUND/LITERATURE REVIEW

Research Paper used for our Project:

1. Metrics for assessing a software system's maintainability:

This paper focuses on defining metrics for measuring the maintainability of software systems. It discusses how metrics could be combined into a single index of maintainability and further classifies the elements that could influence the maintainability of software systems into three:

- (1) Management practices
- (2) Operational hardware and Software environments
- (3) Target software system

BACKGROUND/LITERATURE REVIEW

CONTINUED...

Research Paper used for our Project:

2. A Practical Model for Measuring Maintainability:

This paper features the limitations in the Maintainability Index (MI) and identifies a number of requirements to be fulfilled by maintainability model to be usable in practice.

The following limitations were identified with MI:-

- Root Cause Analysis

When the MI has a particularly low value, indicating low maintainability, it is not immediately clear what steps can be taken to increase it.

In general, the use of averaging to aggregate measures on individual system parts tends to mask the presence of high-risk parts.

- Average Complexity

- Computability

The Halstead Volume metric, in particular, is difficult to define and to compute.

Reason: There is no consensual definition of what constitutes an operator or an operand in a language such as Java or C#

This paper claims that counting the number of lines of comment, in general, has no relation with maintainability whatsoever. This measure is clearly not solid in MI as it is only an optional measure.

- Comment

BACKGROUND/LITERATURE REVIEW

CONTINUED...

- Understandability

There is no logical argument why the MI formula contains the particular constants, variables, and symbols that it does.

- Control

$$MI = 171 - 5.2 * \ln(V) - 0.23 * (G) - 16.2 * \ln(LOC)$$

Why $16.2 * LOC$ or $0.23 * CC$?

Although having a measure such as the MI at your disposal is obviously more useful than knowing nothing about the state of your systems, the lack of knobs to turn to influence the value makes it less useful as a management tool..

Granularity of source code is studied and more focus on collaborating certain internal attributes is taken into account to reach to some trusted and concrete evidence.

BACKGROUND/LITERATURE REVIEW

CONTINUED...

		source code properties					
		volume	complexity per unit	duplication	unit size	unit testing	
		++	+	-	-	o	
ISO 9126 maintainability	analysability	x		x	x	x	o
	changeability		x	x			-
	stability					x	o
	testability		x		x	x	-

Fig. 5. Mapping source code property scores back to system-level scores for maintainability subcharacteristics. A system-level score is derived for each sub-characteristic by taking a weighted average of the scores of relevant (i.e. marked with a cross) code properties. By default, all weights are equal.

RESEARCH QUESTIONS

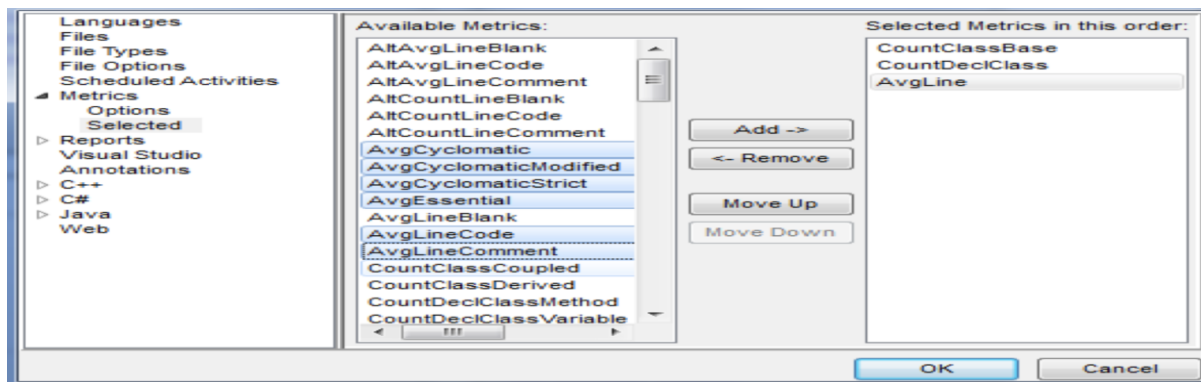
- Maintainability Index is a quantified value, but can maintainability model which takes into consideration the whole of source code produce more significant results?
- Even if it does prove accurate for further analysis, can we validate it with existing comments?
- Is it possible, that a model can show where changes are required to improve or to attribute the cause of reduced maintainability?

RESEARCH METHODS

- Experimental results
- By using tools
 - Understand
 - PMD

UNDERSTAND

- Customizable integrated development environment (IDE)
- Help software developers comprehend, maintain, and document their source code.
- Enables code comprehension by providing flow charts of relationships and
- Build a dictionary of variables and procedures from a provided source code.



PMD

- **Open source static source code analyzer**
- Includes built-in rule sets and supports the ability to write custom rules.
- Does not report compilation errors, as it only can process well-formed source files.

DATA COLLECTION

1. Internal Metrics Implementation

- Volume
- Complexity per Unit
- Unit Size
- Duplication

VOLUME

- The total size of system is very important when it comes to the measure of maintainability.

Metric: - $\text{CountLineCodeDecl} + \text{CountLineCodeExe}$

Two metrics are first calculated based on their availability in Understand tool:

1. ***CountLineCodeDecl*** :- Number of lines containing declarative source code. Note that a line can be declarative and executable - `int i=0;` for instance.
2. ***CountLineCodeExe*** :- Number of lines containing executable source code.

VOLUME CONTINUED....

- **Application in the Project:** These two metrics when combined or summed together for all the files yield us Volume for each version.
- In understand tool, CountLineCode metric gives us a result inclusive of blank lines and comment.

Below table is finally used to categories the result for each version

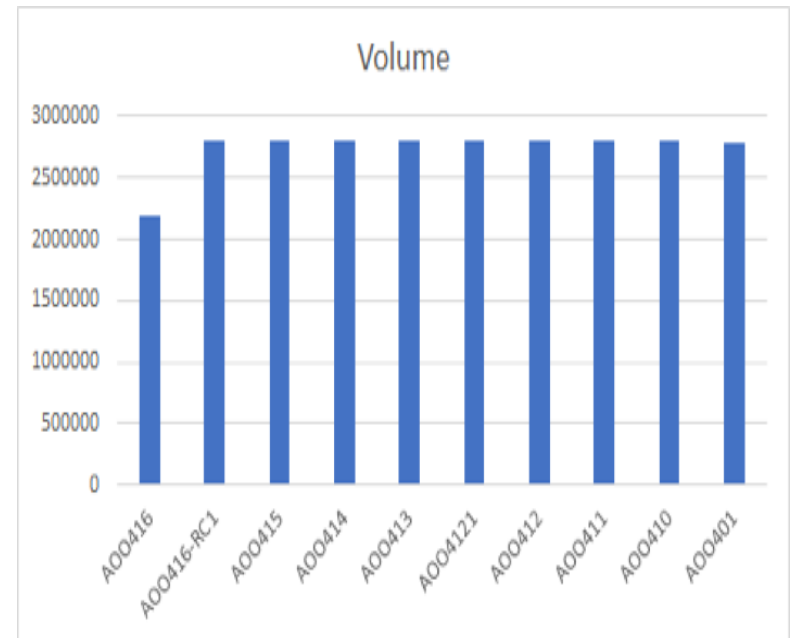
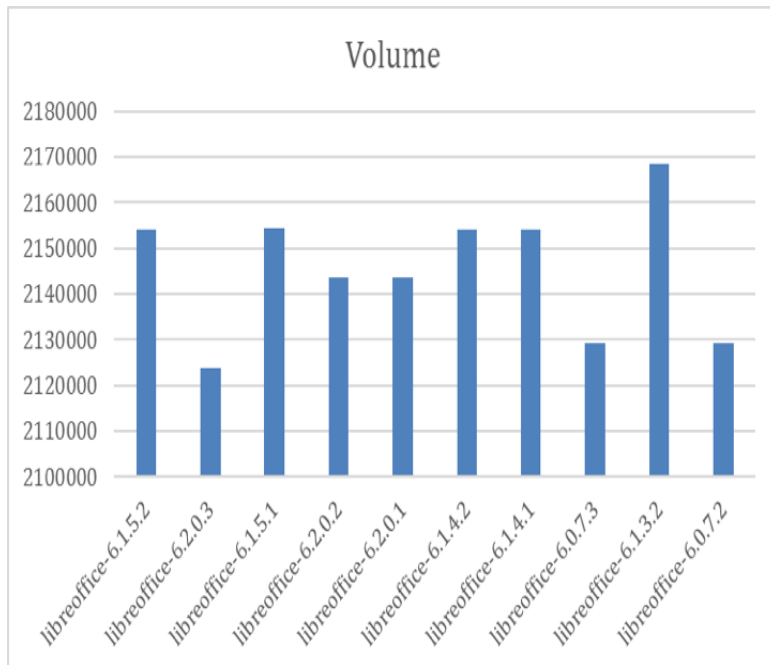
rank	MY	KLOC		
		Java	Cobol	PL/SQL
++	0 – 8	0-66	0-131	0-46
+	8 – 30	66-246	131-491	46-173
o	30 – 80	246-665	491-1,310	173-461
-	80 – 160	655-1,310	1,310-2,621	461-922
—	> 160	> 1,310	> 2,621	> 922

CODE FOR CALCULATING THE VOLUME:

 C:\Users\si_sharm\Downloads\Measurement\rankFunction.R - R Editor

```
rankMRLOC <- function(M, H, VH) {  
  if (M <= 25 && H == 0 && VH == 0) {  
    print('++')  
  } else if (M <= 30 && H <= 5 && VH == 0) {  
    print('+')  
  } else if (M <= 40 && H <= 10 && VH == 0) {  
    print('o')  
  } else if (M <= 50 && H <= 15 && VH <= 5) {  
    print('-')  
  } else {  
    print('--')  
  }  
}
```

ANALYSIS OF LIBREOFFICE AND OPENOFFICE OF VOLUME



COMPLEXITY PER UNIT

Degree of internal intricacy of the source code units

Metric:- AvgCyclomatic

Calculated the average cyclomatic complexity for each unit through the referred tool: -

AvgCyclomatic:- Average cyclomatic complexity for all nested functions or methods.

COMPLEXITY PER UNIT CONTINUED...

Following categorization of units by complexity, provided by the Software Engineering Institute used

CC	Risk evaluation
1-10	simple, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk
> 50	untestable, very high risk

Perform aggregation of complexities per unit by counting for each risk Level what percentage of lines of code falls within units categorized at that level.

STEPS MENTIONED FOR COMPUTING COMPLEXITY

R C:\Users\si_sharm\Downloads\Measurement\CC.R - R Editor

```
source("rankFunction.R")

Data <- read.csv(file.choose())
Data2 <- Data[Data$Kind == "Class",]

#Calculate Sum of Data$AvgCyclomatic

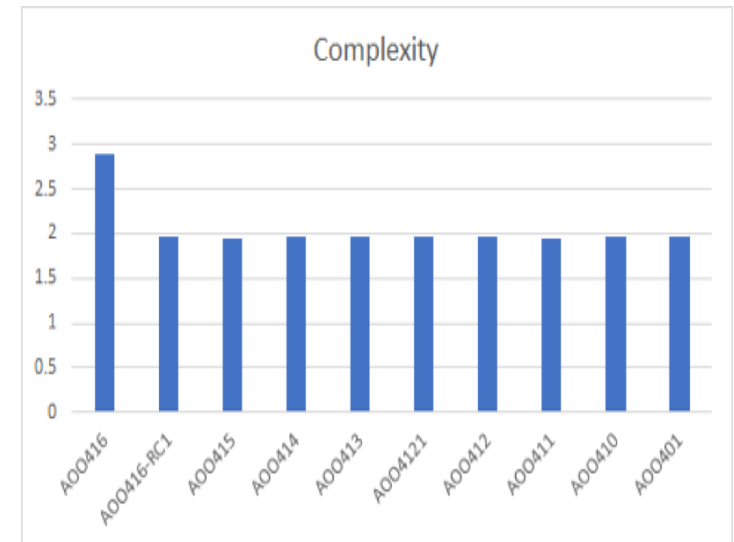
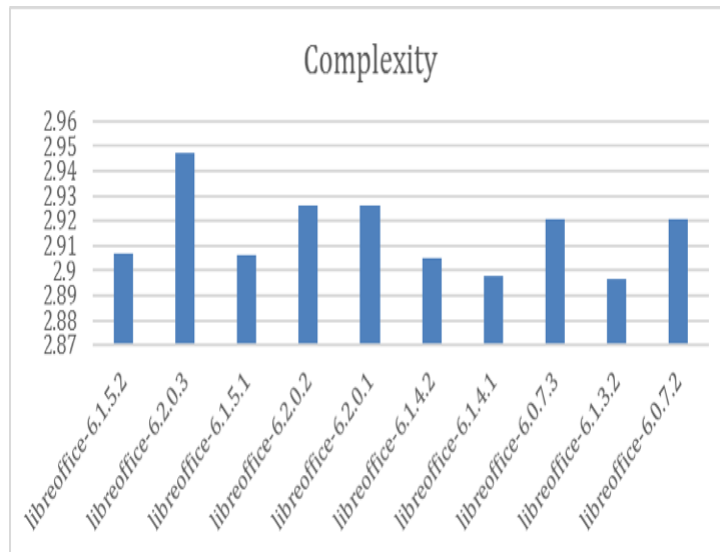
#According to our model we calculate Moderate, High and very High risk
AvgCyclomaticModerate <- Data2[Data2$AvgCyclomatic > 10 & Data2$AvgCyclomatic < 21,]
AvgCyclomaticHigh <- Data2[Data2$AvgCyclomatic > 20 & Data2$AvgCyclomatic < 51,]
AvgCyclomaticVeryHigh <- Data2[Data2$AvgCyclomatic > 50,]

Data2LOC <- sum(Data2$CountLineCodeDecl) + sum(Data2$CountLineCodeExe)
AvgCyclomaticModerateLOC <- sum(AvgCyclomaticModerate$CountLineCodeDecl) + sum(AvgCyclomaticModerate$CountLineCodeExe)
AvgCyclomaticHighLOC <- sum(AvgCyclomaticHigh$CountLineCodeDecl) + sum(AvgCyclomaticHigh$CountLineCodeExe)
AvgCyclomaticVeryHighLOC <- sum(AvgCyclomaticVeryHigh$CountLineCodeDecl) + sum(AvgCyclomaticVeryHigh$CountLineCodeExe)

AvgCyclomaticModeratePercentage <- (AvgCyclomaticModerateLOC/Data2LOC) * 100
AvgCyclomaticHighPercentage <- (AvgCyclomaticHighLOC/Data2LOC) * 100
AvgCyclomaticVeryHighPercentage <- (AvgCyclomaticVeryHighLOC/Data2LOC) * 100
Mean <- (AvgCyclomaticModeratePercentage + AvgCyclomaticHighPercentage + AvgCyclomaticVeryHighPercentage) / 3
Mean

#call rankMRLOC()
rankResult <- rankMRLOC(AvgCyclomaticModeratePercentage, AvgCyclomaticHighPercentage, AvgCyclomaticVeryHighPercentage)
```

ANALYSIS OF LIBREOFFICE AND OPENOFFICE OF COMPLEXITY



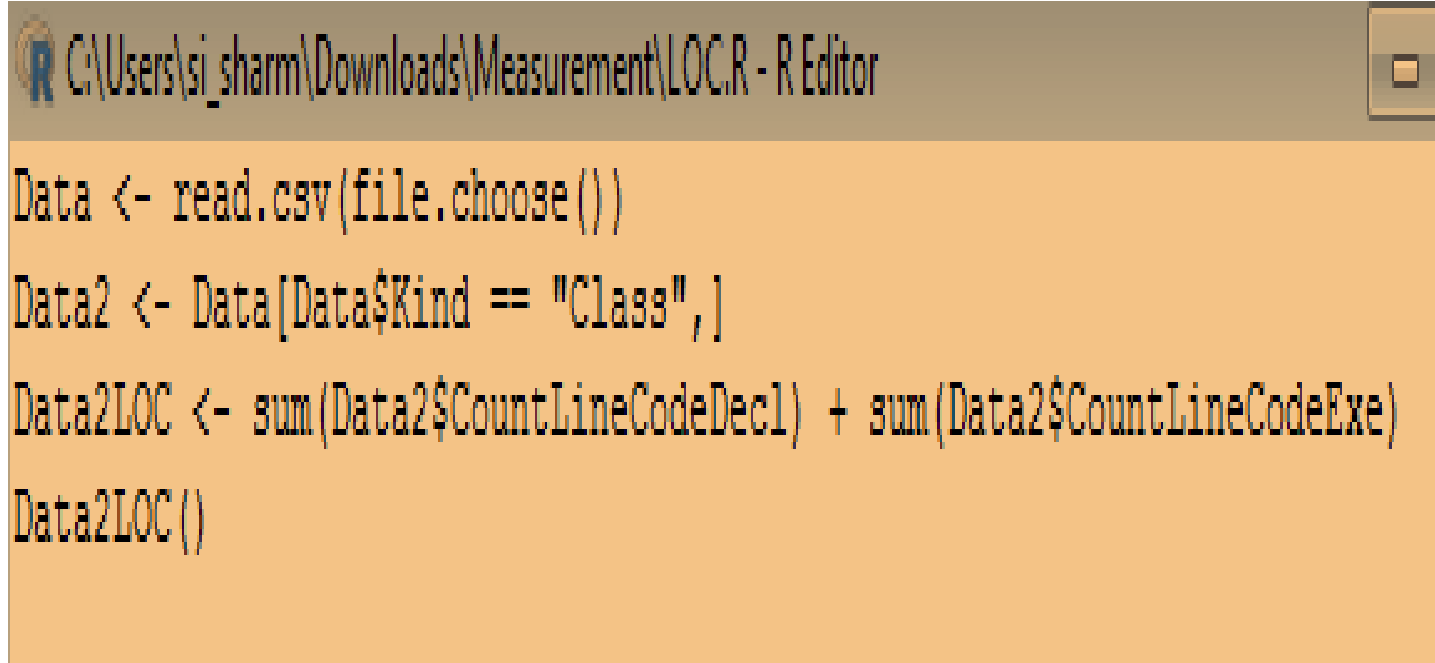
UNIT SIZE

The size of each unit throws light on its maintainability.

Larger unit are difficult to maintain not only because of lower analyzability but lower testability as well.

Metric: - $(\text{CountLineCodeDecl} + \text{CountLineCodeExe})$.

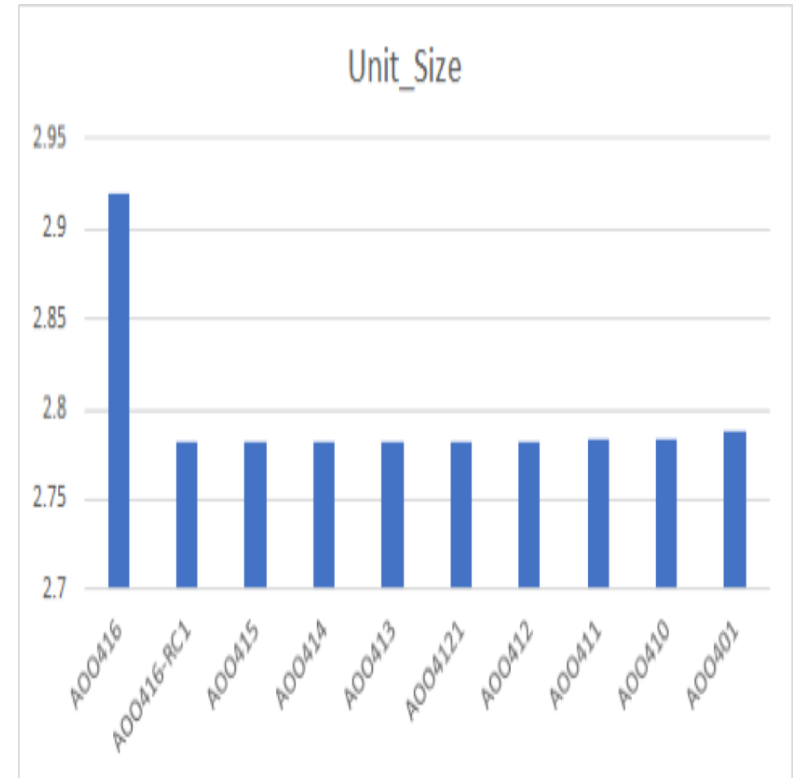
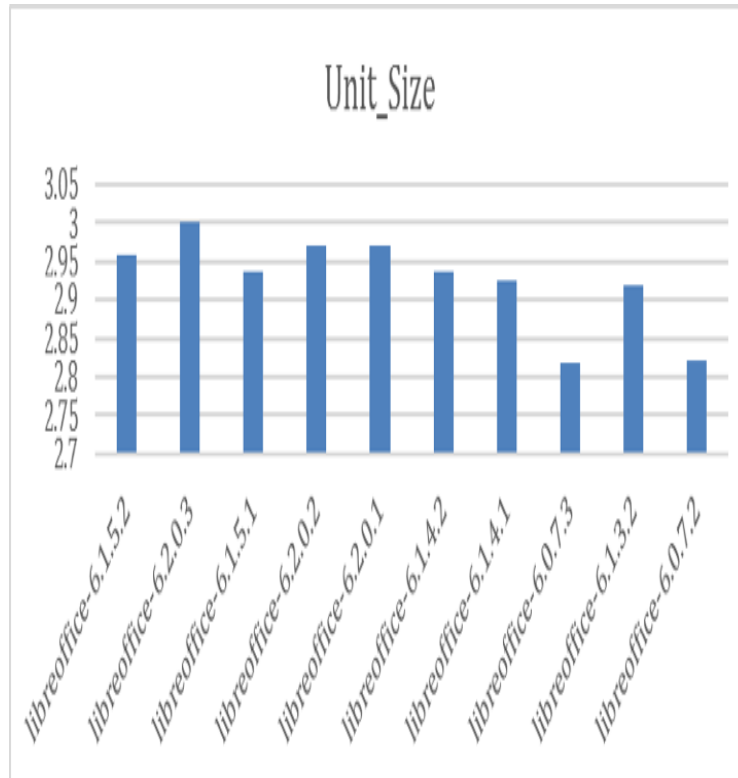
CODE FOR CALCULATING THE UNIT-SIZE



The image shows a screenshot of an R Editor window. The title bar reads "C:\Users\si_sharm\Downloads\Measurement\LOC.R - R Editor". The editor contains the following R code:

```
Data <- read.csv(file.choose())  
Data2 <- Data[Data$Kind == "Class",]  
Data2LOC <- sum(Data2$CountLineCodeDecl) + sum(Data2$CountLineCodeExe)  
Data2LOC()
```

ANALYSIS OF LIBREOFFICE AND OPENOFFICE OF UNIT SIZE



DUPLICATION

Code duplication is the percentage of all code that occurs more than one in equal code blocks at least 6 lines. Leading spaces are ignored.

Metric: - Duplicated Lines of Code

Application in the Project: - Once the count of duplicate code is observed and finally the percentage is determined in terms of the whole source code.

rank	duplication
++	0-3%
+	3-5%
o	5-10%
-	10-20%
--	20-100%

CODE FOR CALCULATING THE DUPLICATION

```
R C:\Users\sj_sharm\Downloads\Measurement\UnitSize.R - R Editor
source("rankFunction.R")

Data <- read.csv(file.choose())
Data2 <- Data[Data$Kind == "Class",]

#Calculate Sum of Data$AvgunitSize

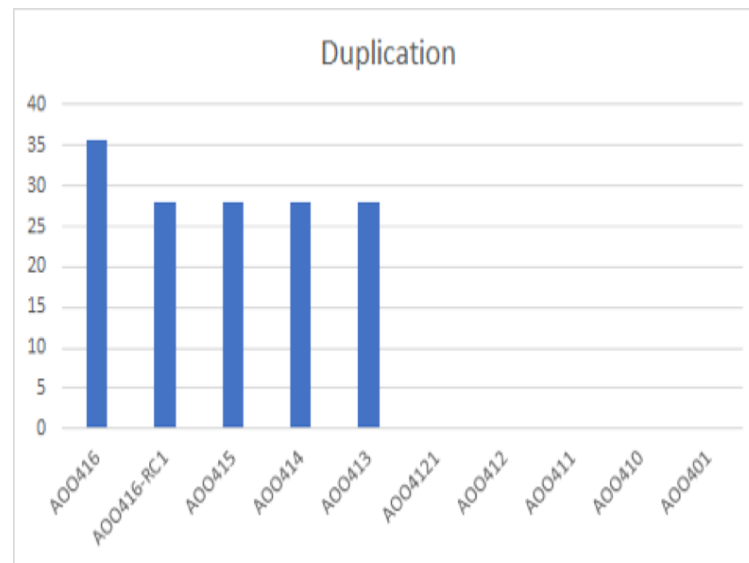
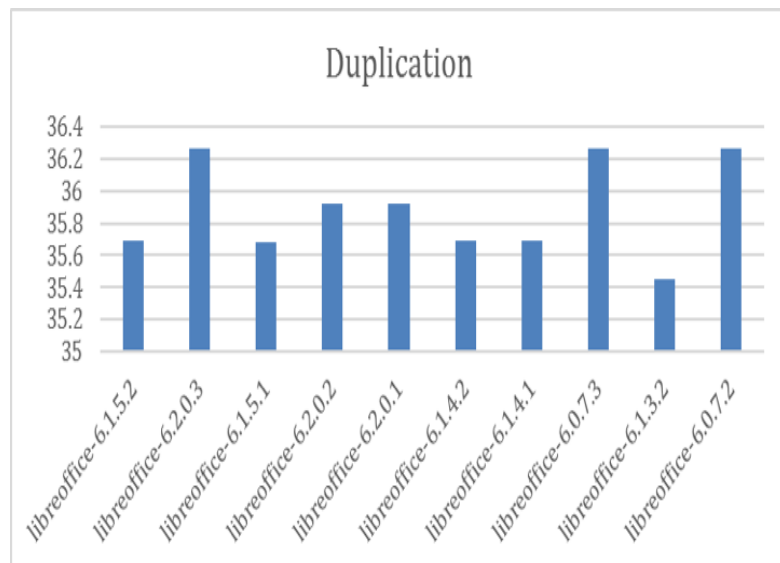
#According to our model we calculate Moderate, High and very High risk
AvgUnitModerate <- Data2[Data2$CountLineCodeDecl < 501 && Data2$CountLineCodeDecl > 301 && Data2$CountLineCodeExe < 1000,]
AvgUnitHigh <- Data2[Data2$CountLineCodeDecl < 1000 && Data2$CountLineCodeDecl > 500 && Data2$CountLineCodeExe < 1000,]
AvgUnitVeryHigh <- Data2[Data2$CountLineCodeDecl > 1000 & Data2$CountLineCodeExe > 1000,]

Data2LOC <- sum(Data2$CountLineCodeDecl) + sum(Data2$CountLineCodeExe)
AvgUnitModerateLOC <- sum(AvgUnitModerate$CountLineCodeDecl) + sum(AvgUnitModerate$CountLineCodeExe)
AvgCyclomaticHighLOC <- sum(AvgUnitHigh$CountLineCodeDecl) + sum(AvgUnitHigh$CountLineCodeExe)
AvgUnitVeryHighLOC <- sum(AvgUnitVeryHigh$CountLineCodeDecl) + sum(AvgUnitVeryHigh$CountLineCodeExe)

AvgUnitModeratePercentage <- (AvgUnitModerateLOC/Data2LOC) * 100
AvgUnitHighPercentage <- (AvgCyclomaticHighLOC/Data2LOC) * 100
AvgUnitVeryHighPercentage <- (AvgUnitVeryHighLOC/Data2LOC) * 100
Mean <- (AvgUnitModeratePercentage + AvgUnitHighPercentage + AvgUnitVeryHighPercentage ) /3
Mean

#call rankMRLOC()
rankResult <- rankMRLOC(AvgUnitModeratePercentage, AvgUnitHighPercentage, AvgUnitVeryHighPercentage)
```

ANALYSIS OF LIBREOFFICE AND OPENOFFICE OF DUPLICATION



CHALLENGES FACED DURING METRICS IMPLEMENTATION

- Plugin suggested by Understand tool and it did work but only limited to certain size of files.
- To overcome this, we came up with the PMD tool, which helped us count the number of duplicated lines of code in a million source code lines.
- An Amazon EC2 instance with 32 Gig of RAM and finally ran the PMD tool for different versions.

DIFFICULTY FACED USING UNDERSTAND

Understand

- The names and definitions with regards to metrics, when compared between the research paper and the tool had inadequacy and ambiguity at certain places.

Overcame this difficulty by defining the concrete definition in all the aspects and then finding the corresponding metrics to it.

DIFFICULTY FACED USING UNDERSTAND CONTINUED.....

Volume

Volume is determined through LOC that excludes comment or blank lines.

To **Overcome** this difficulty, we calculated the mentioned volume through the calculation of two separate metrics and then adding them up to find the real result.

Finally, this can be plotted against the mentioned or derived table which will further help us in our analysis.

DIFFICULTY FACED USING UNDERSTAND CONTINUED.....

Unit Size

The only difference between unit size and volume is the unit. In volume the whole source code is into the picture whereas in unit size, individual unit is taken into consideration.

Again **Overcome** by calculating individual metrics at the unit level and then summing them up.

CONCLUSION AND FUTURE RESEARCH

We analyzed the evolution of maintainability in two large scale open source systems.

The proposed Maintainability Model and the Maintainability index were both measure the evolution of both systems across 10 most recent versions.

The generated graphs for the distinct metrics chosen to determine maintainability displayed a **relative stagnancy** in the maintainability of both system. The maintainability model provides knob-turning information relevant to improve the software system.

CONCLUSION AND FUTURE RESEARCH

We think that more systems in production environments may need to be inspected with the proposed maintainability model to find to extent to which it helps to improve various types of systems.

We also think that concrete requirements for the proposed maintainability model would need to be provided to reduce the limitation of expert opinions to allow the accessibility when compared to the MI.

REFERENCES

- [1] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability", in Proceedings Conference on Software Maintenance 1992, Orlando, FL, USA, USA, 1992, pp. 337 - 344.
- [2] I. Heitlager, T. Kuipers and J. Visser, "A Practical Model for Measuring Maintainability", in 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), Lisbon, Portugal, 2007.
- [3] S. R. Chidamber , C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, June 1994.
- [4] GitHub - apache/openoffice: Mirror of Apache OpenOffice", GitHub, 2019. [Online]. Available: <https://github.com/apache/openoffice>. [Accessed: 02- Feb- 2019].
- [5] LibreOffice", GitHub, 2019. [Online]. Available: <https://github.com/LibreOffice>. [Accessed: 01- Feb- 2019]

QUESTIONS?