Project

Sree Kavya Ganja
20000217-T122
srga20@student.bth.se

Prudhvi Nath Naidu Cherukuri
19981212-5533
prch20@student.bth.se

Jeevan Masavarapu
20000204-T077
jems20@student.bth.se

VVSS Varma
19990310-T195
vemu20@student.bth.se

***Abstract*—The main aim of this paper is to evaluate maintainability of an OO system ArtOfIllusion built in java using a GQM based approach. Based on the selected internal attributes (code structure,complexity, understandability and size) relevant OO metrics have been chosen for the analysis. OO metric tool named "Code MR" has been used in evaluation of the internal attributes and maintainability of the system. The metric values of the internal attributes have been taken and analyzed to obtain the results.**

***Index Terms*—Maintainability, Object Oriented metrics, Understandability, Coupling, Cohesion, Complexity.**

I. Introduction

Software development involves many people and different processes. In the present day, we as developers give utmost importance to the customers so the requirements keep changing. Managers and developers are supposed to make different important decisions for the proper development of the product, in such cases metrics are used. Metrics are used to measure the attributes of entities [7]. Using metrics helps us improve and understand what can be done to improve system maintainability, understandability, overall quality of the system and helps in making different important decisions in the software. There are different measures which can be used for measurement but in this study, we mainly focus on OO metrics.

In the present-day object-oriented design is widely used as it is easy to use and gives a higher quality software. Though it is simple, there is a chance for failure if decisions are not properly made and the system is highly complex. In such cases, OO metrics are used to measure the processes and analyze them to take important decisions. These metrics help us make decisions related to budget, release of the product and requirements of the product etc., The main measures in these metrics are based on methods, classes, coupling, cohesion, size, and complexity. From our knowledge from the review assignment the most widely used metrics are Chidamber and Kemerer metrics and Li and Henry metrics. If the complexity of the system is high, then the system is not easily maintainable and understandable. A reasonable complexity rate can be achieved by low coupling and high cohesion. If the modules or the components of the software are loosely coupled and the interconnections between methods in the class are highly cohesive then the system is easily maintainable.

This report deals with evaluating the maintainability of ArtOfIllusion which is built in java. We have studied seven different releases of the product for better understanding of the product and the development of the product over the years. Maintainability is an external attribute that depends on the main four internal attributes namely Code structure, complexity, understandability, and size. We have used the GQM approach where we have formulated strong questions for each attribute and selected the suitable metrics that will be used in the evaluation from the tools. The main goal of this report is to identify the modules and classes that are difficult to maintain.

The report is further divided into sections as follows: in section II we explain about the research methodology we choose to use in this project. Section III explains about the goal and questions related to GQM tree including relevant justifications and limitations for selected metrics. Section IV the main discussion is about the results and the analysis of the collected data. In Section V we discuss about the related work and reflections from the study.

II. Research Methodology

As the study is an empirical study, we choose case study as the research methodology in this project. The case here is evaluating the maintainability of ArtOfIllusion modules and identify the modules that are difficult to maintain. The reason behind choosing case study as our approach because we are going to have a detailed study of the product first and then analyze the results obtained from the evaluation. First, we closely study the system over seven releases and gather data related to our GQM questions and then try to answer or analyze the data into meaningful solutions and eventually identify the modules that are difficult to maintain.

A. How we collect data?

The main goal of this study is to find out the modules that are difficult to maintain. In order to evaluate maintainability we need to extract the metrics related to the internal attributes. To extract these metrics we have use the tool "Code MR" in this project. Code MR is a metric extraction tool which is integrated with eclipse and IntelliJ IDEA. Code MR is easy to use and visualize our project. we have also tried to use metrics reloaded in our project but it does not have all the metrics that

we have selected for analysis. Considering the simplicity and availability of the relevant metrics we have chosen Code MR as the main tool.

Code MR will generate metrics values based on the selected project in the IDE. It will show the values that are supported within the project that we are working, we just need to upload the project and select the modules that we would like to extract the metrics. We need to right click on the selected module from the given options select analyse and then select calculate metrics. Now, for the selected module the supported metrics that can be extracted by the code MR will be available to us in many different formats like package structural format, detailed metrics list, metric explanation and related graphs can be obtained.

B. Analysis

From the generated data, the data is at package level. For some metrics the tool is providing only at class level we have aggregated these class level metrics to package level by calculating the sum of all the elements in that package and considered it as a whole. We have used bar graphs and line charts to represent the data visually for analysis of seven versions of ArtOfIllusion. we have used some graphs generated from tool and few of them from excel. From the graphs we were able to tell the evolution of different metrics and how they affect the code structure, understandability, and overall product. In this way we have analysed the data and produced results.

III. GQM TREE

Entity: ArtOfIllusion (modules)
Internal Attributes: code size, code understandability, code complexity, code structure (coupling, cohesion).
External Attributes: Maintainability

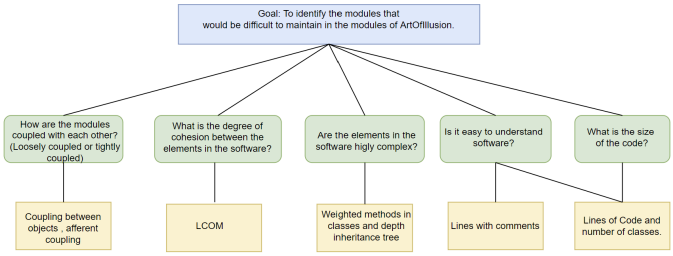

Fig. 1. GQM Tree

A. Description, justification and Limitations of metrics

1) ***CBO (Coupling between objects):*** This is a coupling metric and it is used to calculate the total number of classes on which the class is dependent on i.e., if two are objects are coupled with each other then one object can use the methods of the other object. But highly coupled classes are fault prone as they are dependent on class methods and these classes are also not reusable. This metric was widely used since it is mentioned by Chidamber and Kemerar OO metrics. [5]
Justification: Finding out coupling in between objects is very important because high coupling causes harm to overall design and it is also not reliable as it can lead to many errors. it has also been used in many studies that have used and highlighted the importance of CBO. In [6] it is said that CBO is connected with the defect occurrence in the system and in different studies like [8] [9], [11], [17] have used CBO to detect changes, and predicting maintainability and it is also said to be a mature metric in terms of reliability.
In the current project, we will get CBO values on a module level and we have aggregated those values to a package level. Based on these research papers and from our experience we have considered CBO as an appropriate metric to evaulate coupling in this project.
Limitations: This metric however does not take shared variables into consideration and it is also a class level metric. However, if the coupling value is high then that particular class is not reusable, maintainable, and understandable. So, it is always suggested that the class should not be tightly coupled for a better quality product.
Scale: Ratio.

2) ***Afferent Coupling:*** Another coupling metric which measures the total number of external classes that are coupled to the class of a package because of incoming coupling between the packages.
Justification: This is a package level metric, and it is related to stability. Stability is then correlated to maintainability so, if afferent coupling increases maintainability and reusability is increased. This also works with all the instances regardless of the language used in the code.
Limitations: High afferent coupling results in negative influence of stability, portability and fault proneness.
Scale: Ratio

3) ***Lack of Cohesion between Methods(LCOM):*** This is a cohesion metric and it measures the inter relatability of a class i.e., counts the number of method pairs that are not connected in a class. High LCOM value means low cohesion. It is also mentioned in the CK metrics list and it has been widely used since it is mentioned by Chidamber and Kemerar.
Justification: If the methods in the class are interrelated with each other then it is said to be a cohesive class and such classes have a high chance of generating errors because of the shared variables. This metric helps us understand the cohesion values between the methods and take necessary precautions.
This is also one of the most used metrics from the CK metric suite for measuring cohesion. Lower the LCOM value higher maintainability. High cohesion supports encapsulation indicating high testing efforts. It was used in different studies and it is said to be connected to testability and has been used in prediction studies [4] [10].

Limitations: Its scope is limited to class only. And the results from these values are highly complex. This cannot be used in special cases like not having any attributes at all [2]. Another limitation is that if the variables are privately declared we might not get accurate results.
Scale: Ratio as numerical values and zero exists and calculations like can be used.

*4) **Weighted methods per class(WMC):*** WMC is a complexity metric that calculates the sum of weights of all class methods.High WMC, High complexity. This metric is also part of the list of CK metrics and has been widely used in different studies.
Justification: As we will calculate all the elements of the class like methods, children, objects etc., all these elements together constitute the complexity of the code so we chose this as an appropriate metric for measuring complexity. This can be used for early prediction of maintainability and testability. WMC can also detect faults. This does not provide repetitive information. If there are a greater number of methods defined in a class then there is a huge impact on all the children classes and it also leads to many faults so WMC is an appropriate metric for measuring complexity.
It has been used in different studies to detect code changes in [8] [9], [17]
Limitations: The scope of this metric is limited to class level and it will not measure the parent functions or operators in a child class.This metric actually measures two different attributes the methods in the class and the complexity so the results might be a bit confusing.
Scale: Ratio- as numerical values and zero exists and calculations like mean can be used.

*5) **Lines Of Code(LOC):*** LOC is a size metric and it measures Total number of lines in a code including comments but whitespaces are not counted. It is basically a size measure, but it also measures the maintenance.(Higher LOC, High Complexity, Low maintainablity).
Justification: A good software design is said to have less number of lines compared to a bad software design. in our opinion this is an easiest and intuitive metric that can be used to measure the size. This metric is widely used for measuring complexity, size and change prediction models. [14], [8], [2].
Limitations:: This a language dependent metric for example an application is written using java and other is written using .net then the lines of code will automatically be different. It is also partly dependent on the programmers' experience as a programmer with more experience will use shorter logics and entry level programmers might use a longer approach.
Scale: Ratio- as numerical values and zero exists and calculations like mean can be used.

*6) **Depth Inheritance tree(DIT):*** DIT is another code metric that we choose to measure complexity. It measures the maximum distance from a node to root node in a class. If we consider three classes where class 3 is a child of class 2 and class 2 is a child of class 1. Then, the DIT of class 1 is 0 and 1,2 are the DIT values of the respective classes. [1]
Justification: Higher number in DIT values means high complexity. DIT has been used in numerous studies for detecting change proneness, maintainability [8], [9]and it is also mentioned that it has been used in various studies in [16] to measure complexity.
considering vast number of studies using this metric and from our review assignment it is said to be a reliable metric so, we decided to choose DIT to measure complexity.
Limitations: Its scope is only till class level and it is said to be ambiguous. This metric is highly influenced by the design [18] .
Scale: Ratio- as numerical values and zero exists and calculations like mean can be used.

*7) **Number of Classes(NOC):*** We have considered NOC to be size metric and understandability but it can also used as a complexity metric. It measures the total number of subclasses in a class.
Justification: higher numberof classes higher reusability and it has been widely used in different studies to measure complexity, size and change prediction and also maintainability. [8] [9], [11], [17] [16]. It is also part of CK metrics list so, it is highly reliable to give accurate results.
Limitations: It does not show all the sub classes of the parent class and it does not show interfaces.
Scale: Ratio- as numerical values and zero exists and calculations like mean can be used.

*8) **Lines with Comments:*** lines with comments is an understandability metric. more number of comments in the code higher readability and it is easy to understand. this metric measures all the lines in code with comments.
Justification: when comments are properly written in a program, it easy for the reader to understand the program easily. This increases understandability. So, we choose lines with comments as an appropriate metric for understandability.
Limitations: It measures all the lines with comments. The problem arises when comments are not written and it is just random white spaces.
Scale: Ratio- as numerical values and zero exists and calculations like mean can be used.

IV. Results

A. Section 4.1:

In the latest release of ArtofIllusion 3.2.0 project there are about 15 modules and each module contains several classes that can be observer in the given image named Overview of ArtofIllusion 3.2.0. The project is in the Java language that is identified in the metric extraction process. There modules are named as follows, artofillusion, animation, animation.distortion, image, image.filter, keystroke, material, math, object, procedural, script, texture, UI, unwrap, util, and view named modules are identified in the latest release of

ArtofIllusion project.

Packages of Art of Illusion 3.2.0	LOC	#C
<Package>artofillusion.animation	9062	55
<Package>artofillusion.animation.distortion	2140	18
<Package>artofillusion.image	3231	21
<Package>artofillusion.image.filter	787	10
<Package>artofillusion.keystroke	314	5
<Package>artofillusion.material	701	9
<Package>artofillusion.math	1836	15
<Package>artofillusion.object	11189	50
<Package>artofillusion.procedural	7012	76
<Package>artofillusion.script	962	14
<Package>artofillusion.texture	5069	42
<Package>artofillusion.ui	3675	52
<Package>artofillusion.unwrap	312	2
<Package>artofillusion.util	1297	6
<Package>artofillusion.view	2856	24
Average of all packages	3362.866667	26.6

Fig. 2. Overview of ArtofIllusion 3.2.0

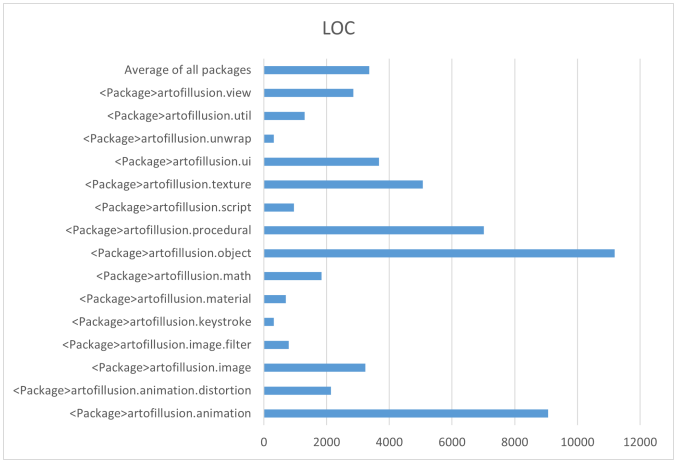

Fig. 3. Line of Code in each module of 3.2.0

From the images of LOC and No of Classes we can see that the avg size of the version 3.2.0 according to LOC is 4563.5 and the avg size according to the No of Classes is 31. The outliers of the project as follows, the module that contains the highest LOC is module artofillusion and the lowest LOC is module artofillusion.unwrap. The outlier modules according to the No of Classes are, the highest classes containing module with 97 classes is artofillusion and the lowest classes containing module with 2 classes is artofillusion.unwrap.

The main classes that are identified in each module are listed in the table named Main classes in each module of 3.2.0. This table gives an overview of what are the main classes in every module,this evaluation of main classes is based on the values of LOC and NOM of each module. We have considered the highest value of LOC and NOM to identify the main class in each module. The avg value of LOC and NOM are also calculated and provided in the table.

After the evaluation of all 7 versions of ArtofIllusion project in terms of size such as LOC, No of classes in each module across 7 versions we have gathered the data regarding the LOC

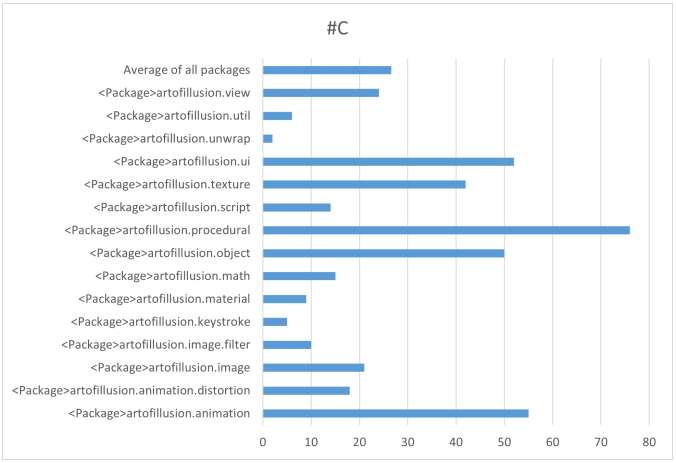

Fig. 4. No of classes in each module of 3.2.0

Packages of Art of Illusion 3.2.0			
	Class	LOC	NOM
<Package>artofillusion.object	artofillusion.object.TriangleMesh	2343	47
<Package>artofillusion.animation	artofillusion.animation.Score	705	57
<Package>artofillusion.procedural	artofillusion.procedural.ProcedureEditor	616	16
<Package>artofillusion.texture	artofillusion.texture.ProjectionMapping	419	24
<Package>artofillusion.ui	artofillusion.ui.Compound3DManipulator	783	12
<Package>artofillusion.image	artofillusion.image.ImagesDialog	534	12
<Package>artofillusion.view	artofillusion.view.SoftwareCanvasDrawer	1230	9
<Package>artofillusion.animation.distortion	animation.distortion.CustomDistortionTrack	290	25
<Package>artofillusion.math	artofillusion.math.SimplexNoise	456	0
<Package>artofillusion.util	artofillusion.util.IconGenerator	992	1
<Package>artofillusion.script	artofillusion.script.ScriptedObject	258	14
<Package>artofillusion.image.filter	artofillusion.image.filter.OutlineFilter	185	10
<Package>artofillusion.material	artofillusion.material.ProceduralMaterial3D	227	16
<Package>artofillusion.keystroke	keystroke.KeystrokePreferencesPanel	112	7
<Package>artofillusion.unwrap	artofillusion.unwrap.SeamFinder	312	7
Average of all packages		630.8	17.1333

Fig. 5. Main classes in each module of 3.2.0

LOC of all Versions								
Package Names	Versions							
	3.0.3	3.1.0	3.2.0	N-6d119b1	N-47c3f74	N-187a2b3	N-2020-06-25	Avg Value
<Package>artofillusion.animation	8922	8919	9062	8919	8919	8924	9057	8960.285714
<Package>artofillusion.animation.distortion	2138	2140	2140	2140	2140	2142	2140	2140
<Package>artofillusion.image	1997	3258	3231	3259	3262	1997	3258	2894.571429
<Package>artofillusion.image.filter	787	787	787	787	787	787	787	787
<Package>artofillusion.keystroke	304	304	314	304	304	304	304	305.4285714
<Package>artofillusion.material	698	701	701	698	701	698	701	699.7142857
<Package>artofillusion.math	1773	1836	1836	1836	1836	1826	1836	1825.571429
<Package>artofillusion.object	11112	11143	11189	11130	11134	11129	11189	11146.57143
<Package>artofillusion.procedural	7027	7037	7012	7031	7037	7027	7012	7026.142857
<Package>artofillusion.script	956	956	962	956	956	956	956	956.8571429
<Package>artofillusion.texture	5067	5069	5069	5068	5069	5068	5069	5068.428571
<Package>artofillusion.ui	3498	3541	3675	3521	3540	3511	3674	3565.714286
<Package>artofillusion.unwrap	312	312	312	312	312	312	312	312
<Package>artofillusion.util	1296	1297	1297	1297	1297	1296	1297	1296.714286
<Package>artofillusion.view	2352	2829	2856	2770	2834	2766	2856	2751.857143

Fig. 6. LOC of all modules from 7 versions of ArtofIllusion

No of Classes of all Versions								
Package Names	Versions							
	3.0.3	3.1.0	3.2.0	N-6d119b1	N-47c3f74	N-187a2b3	N-2020-06-25	Avg Value
<Package>artofillusion.animation	55	55	55	55	55	55	55	55
<Package>artofillusion.animation.distortion	18	18	18	18	18	18	18	18
<Package>artofillusion.image	17	21	21	17	21	17	21	19.28571429
<Package>artofillusion.image.filter	10	10	10	10	10	10	10	10
<Package>artofillusion.keystroke	5	5	5	5	5	5	5	5
<Package>artofillusion.material	9	9	9	9	9	9	9	9
<Package>artofillusion.math	14	15	15	15	15	15	15	14.85714286
<Package>artofillusion.object	50	50	50	50	50	50	50	50
<Package>artofillusion.procedural	75	76	75	75	76	75	76	75.42857143
<Package>artofillusion.script	15	15	15	15	15	15	15	15
<Package>artofillusion.texture	42	42	42	42	42	42	42	42
<Package>artofillusion.ui	52	52	52	52	52	52	52	52
<Package>artofillusion.unwrap	2	2	2	2	2	2	2	2
<Package>artofillusion.util	6	6	6	6	6	6	6	6
<Package>artofillusion.view	18	23	22	22	23	22	24	22

Fig. 7. NOC of all modules from 7 versions of ArtofIllusion

and No of Classes metrics. The avg values of LOC and No of Classes of all the modules were very close to the actual value of all the 7 versions ANd we have identified that the no of packages is same across all the 7 versions of ArtofIllusion. We can see the details of all the modules from the above images regarding the LOC and NOC of all versions.

B. Section 4.2:

How are the modules coupled with each other(loosely or tightly coupled)?

The evaluation of coupling metrics across all the seven versions of ArtOfIllusion product is based on the two selected metrics such as Afferent coupling(AC) and coupling between objects(CBO). Afferent coupling is a package level metric so we did not have to aggregate the values whereas CBO is a class level metric and its values have been aggregated to package level. We extracted both the metric values of all the modules across seven versions. Higher the coupling value higher the chance of having defects and difficult to maintain. From the obtained results of coupling metrics it is identified that modules named as artofillusion, artofillusion.animation, artofillusion.object, and artofillution.UI are the highly coupled modules whereas the modules that are loosely coupled are image, image.filter, keystroke, material, script, unwrap, util, and view packages.

Coupling Metrics					
Package Names		AC		CBO	
<Package>artofillusion.animation		55.2857	Medium	502.2857	High
<Package>artofillusion.animation.distortion		4	Low	191.1429	Medium
<Package>artofillusion.image		27.8571	Low	67.57143	Low
<Package>artofillusion.image.filter		5	Low	57	Low
<Package>artofillusion.keystroke		0.85714	Low	22.28571	
<Package>artofillusion.material		21	Low	62.42857	Low
<Package>artofillusion.math		243.286	High	21.71429	Low
<Package>artofillusion.object		155	Medium	436.4286	Medium
<Package>artofillusion.procedural		10	Low	316.1429	Medium
<Package>artofillusion.script		2.85714	Low	54.14286	Low
<Package>artofillusion.texture		49.1429	Low	361.2857	Medium
<Package>artofillusion.ui		161.286	Medium	125.8571	Medium
<Package>artofillusion.unwrap		0	Low	7	Low
<Package>artofillusion.util		2	Low	0	Low
<Package>artofillusion.view		8.57143	Low	93.14286	Low
Scale for AC	Low: <50; Medium 50-200; High: >200				
Scale for CBO	Low: <100; Medium: 100-500; High: >500				

Fig. 8. Coupling Metric Values

What is the degree of cohesion between the elements in the software?

Cohesion is the degree of ”binding” between elements in a software unit (module). A higher degree of cohesion is desired because that indicates that the elements are tightly bound and as such ensure robustness, reusability, and understandability. Lack of Cohesion Of Methods (LCOM) is the metric being used in this scenario. LCOM is, as the name indicates, the lack of cohesion between methods and is measured as the attributes that are used in common between methods. Higher cohesion, and thereby LCOM ensures a higher maintainability.

The LCOM values have been averaged across the seven versions of the software and the value thus obtained has been

Cohesion Metrics		
Package Names	LCOM	
<Package>artofillusion.animation	22.2124	Medium
<Package>artofillusion.animation.distortion	6.379	Medium
<Package>artofillusion.image	12.036	Medium
<Package>artofillusion.image.filter	0	Low
<Package>artofillusion.keystroke	1.98214	Low
<Package>artofillusion.material	3.19443	Low
<Package>artofillusion.math	4.03143	Low
<Package>artofillusion.object	20.1766	Medium
<Package>artofillusion.procedural	39.6717	High
<Package>artofillusion.script	2.187	Low
<Package>artofillusion.texture	14.7377	Medium
<Package>artofillusion.ui	13.4384	Medium
<Package>artofillusion.unwrap	0.625	Low
<Package>artofillusion.util	1.547	Low
<Package>artofillusion.view	5.26914	Medium
Scale for LCOM	Low: <5; Medium:5-25, High: >25	

Fig. 9. Cohesion Metric Values

binned into three categories Low, medium, and high. Modules unwrap, util, math, material, keystroke, script, and image.filter have low values of LCOM. Meanwhile, the other modules have a relatively higher values of LCOM.

Are the elements in the software highly complex?

Complexity is a term that indicates how difficult or easy it is for a software element to be understood, managed, or modified. Lower complexity is desirable as it means that the software element can be easily reused, replaced, or modified. Weighted Method per Class (WMC) and Depth of Inheritance Tree (DIT) are the metrics being used to judge the complexity of modules. Lower values of DIT and WMC are desirable because they indicate a lower complexity. Here, the DIT values of each package are taken and their average across all seven versions is calculated. That value is then binned into three categories: Low, medium, and high. Same is done with WMC. Then, we take both the values of WMC and DIT for each package into consideration to draw conclusions. The values of DIT and WMC are relatively higher in animation, procedural, object, and ui. These modules have a higher complexity. util, and unwrap have very low complexities. image.filter, keystroke, material, math, and view also have below average complexities. The rest of the modules have complexities that are a slightly on the higher side.

Is it easy to understand the software?

The evaluation of understanability metrics is performed using CLOC metric and size metric NOC and LOC. NOC and LOC are also considered as understandability metrics because, these metrics also tell us about understandability. for example more number of lines means it is difficult to understand similarly higher number of classes means low understandability. We did not obtain the values of CLOC metric at package level so, we have extracted the metrics at both class level and method level. However, the values obtained did not show a huge difference among all the versions of CLOC metric. This

Complexity Metrics				
Package Names	DIT		WMC	
<Package>artofillusion.animation.distortion	21.2857	Medium	454.143	Medium
<Package>artofillusion.image	10.8571	Medium	419.286	Medium
<Package>artofillusion.image.filter	9	Low	204	Medium
<Package>artofillusion.keystroke	7.71429	Low	41.8571	Low
<Package>artofillusion.material	6	Low	105.286	Medium
<Package>artofillusion.math	0	Low	427.143	Medium
<Package>artofillusion.object	36	Medium	2169.57	High
<Package>artofillusion.procedural	93.1429	High	1674.57	High
<Package>artofillusion.script	17	Medium	162.286	Medium
<Package>artofillusion.texture	46.8571	Medium	1029.29	Medium
<Package>artofillusion.ui	59.7143	High	686.286	Medium
<Package>artofillusion.unwrap	0	Low	73	Low
<Package>artofillusion.util	0	Low	49	Low
<Package>artofillusion.view	3	Low	429	Medium
Scale for DIT	Low:<10,Medium:10-50;High:>50			
Scale for WMC	Low: <100; Medium: 100- 1200; High : >1200			

Fig. 10. Complexity metrics values

Understandability Metrics		
Version Name	CLOC in method level	CLOC in Class level
3.0.3	8098	10758
3.1.0	8796	11505
3.2.0	8759	11539
N-6d119b1	8790	11518
N-47c3f74	8819	11554
N-187a2b3	8626	11350
N-2020-06-25	8760	11538

Fig. 11. Understandability Metric Values

indicates that the understandability of the given product is of medium level according to the figure below. According NOC and LOC size metrics understandability is low for few modules like animation, math, object, procedural and high for remaining other modules.

What is the size of the code?

The main metrics used to evaluate size are LOC and NOC. After the extraction of LOC and NOC metric values we have calculate the mean values of all the modules across the seven versions of the product to get the highest and lowest valued metrics. From the table related to size metrics we can classify the module with large code size and the modules with low code size. We observed that animation, procedural and ui modules have more number of classes and modules math, object have more number of LOC.

C. Section 4.3:

In this project the main goal is to identify the modules that are difficult to maintain. Generally, a module is said to be maintainable if the coupling and complexity values are low and a high cohesive value. In our case, to identify the modules that are difficult to main the condition for the modules is to have high complexity, coupling and size and low cohesion value.

Size Metrics				
Package Names	LOC		NOC	
<Package>artofillusion.animation	252.071	Medium	55	High
<Package>artofillusion.animation.distortion	220.286	Medium	18	Medium
<Package>artofillusion.image	107.429	Medium	19.2857	Medium
<Package>artofillusion.image.filter	25.4286	Low	10	Medium
<Package>artofillusion.keystroke	56.5	Low	5	Low
<Package>artofillusion.material	216.571	Medium	9	Low
<Package>artofillusion.math	1084.79	High	14.8571	Medium
<Package>artofillusion.object	855.286	High	50	Medium
<Package>artofillusion.procedural	127.714	Medium	75.4286	High
<Package>artofillusion.script	523.143	Medium	15	Medium
<Package>artofillusion.texture	366.571	Medium	42	Medium
<Package>artofillusion.ui	66.3571	Low	52	High
<Package>artofillusion.unwrap	24.5	Low	2	Low
<Package>artofillusion.util	214.5	Medium	6	Low
<Package>artofillusion.view	3	Low	22	Medium
Scale for LOC	Low: <100; Medium: 100- 600; High : >600			
Scale for NOC	Low:<10,Medium:10-50;High:>50			

Fig. 12. Size metrics values

From our analysis of selected metrics we have identified the modules that are more difficult to maintain in the given project. We have selected the most difficult modules to maintain based on the metrics values of complexity, coupling and size of the code. The modules are as follows:

1) Object Module:: This module is identified as difficult to maintain among the other modules of the given project because this module is having high complexity, coupling, and size of the code. This can be observed from the provided metric values of this module in terms of both tabular form and graph form. From the data it is evident that there is a slight increase in complexity, coupling and size of code metric values. And this can be more clearly observed when we look at the results from the section 4.2 which is all about the selected metric values.

Package - Object								
	Versions							
Metrics	3.0.3	3.1.0	3.2.0	N-6d119b1	N-47c3f74	N-187a2b3	N-2020-06-25	Avg of all
CBO	441	441	409	441	441	441	441	436.4285714
CA	153	156	154	155	155	155	157	155
LCOM	20.134	20.184	20.183	20.184	20.184	20.184	20.183	20.17657143
WMC	2166	2172	2164	2172	2172	2172	2169	2169.571429
DIT	36	36	36	36	36	36	36	36
NOC	50	50	50	50	50	50	50	50
LOC	11112	11143	11189	11130	11134	11129	11189	11146.57143

Fig. 13. Object Module metrics tabular form

2) Procedural Module:: This Procedural module is found in the all the version of the project and is identified as a difficult module to maintain based on the complexity, and size of code metric values. The metrics values states that there is a high metrics values of WMC and DIT metrics and the mean of them can also observe from the given tabular data and graph data. The other metrics is size of code metric where the LOC of the module is very high across all the other versions. A clear explanation is provided in the previous section 4.2 where we have discussed the comparision of all the other attributes.

3) UI Module: This module is also considered as difficult to maintain among all other modules in the project based on

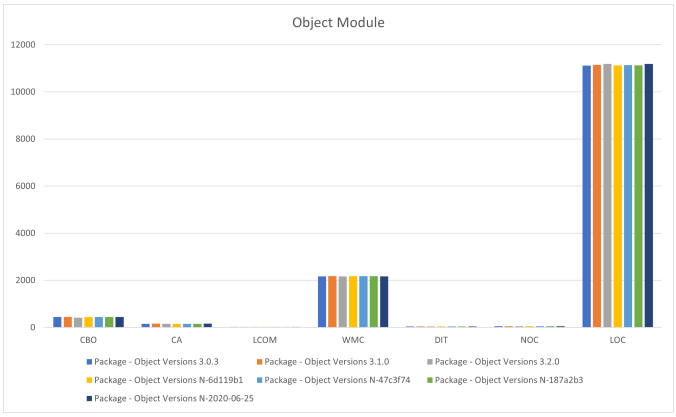

Fig. 14. Object Module metrics graph

Package - Procedural								
	Versions							
Metrics	3.0.3	3.1.0	3.2.0	N-6d119b1	N-47c3f74	N-187a2b3	N-2020-06-25	Avg of all
CBO	291	346	302	291	346	291	346	316.1428571
CA	10	10	10	10	10	10	10	10
LCOM	39.887	39.885	38.382	39.887	39.887	39.887	39.887	39.67171429
WMC	1696	1697	1543	1696	1697	1696	1697	1674.571429
DIT	62	118	112	62	118	62	118	93.14285714
NOC	75	76	75	75	76	75	76	75.42857143
LOC	7027	7037	7012	7031	7037	7027	7012	7026.142857

Fig. 15. Procedural Module metrics tabular form

the complexity metrics, and coupling metrics. We can observe from the below tabular data and graph data of UI module that there is high metrics values of complexity such as WMC and DIT metrics and also high values of coupling metrics such as AC and CBO metrics. These values also not differed to high degree in all the versions of the project. These metrics values are more clearly explained in the section 4.2.

4) Script Module:: This module is felt difficult to maintain from the other modules of artofillusion project because it is having low cohesion and high size of code metrics values that can be observed from the below images. In the tabular data we can see that the cohesion metrics values LCOM are having very low and size metrics values LOC are having high values over all the 7 version of the project and furthrmore when we compare this to the results obtained in section 4.2 we identified this is can also be considered as difficult to maintain module of artofillusion.

5) Texture Module:: This is the less prioritized module among the most difficult to maintain modules as this is having only high complexity metrics and the other metrics are of at optimal level. Here the complexity metrics such as WMC and DIT are having high metric values that can be observed from the provided tabular data below. It is also compare to the results obtained from the section 4.2 about the individual metrics evaluation and analysis.

After all the comparison and analysis of modules across all the 7 version of ArtofIllusion we came to our end conclusion that the most difficult to maintain modules are Oject, Procedural, UI, and Script modules because they are not satisfying the conditions of maintainability standards by

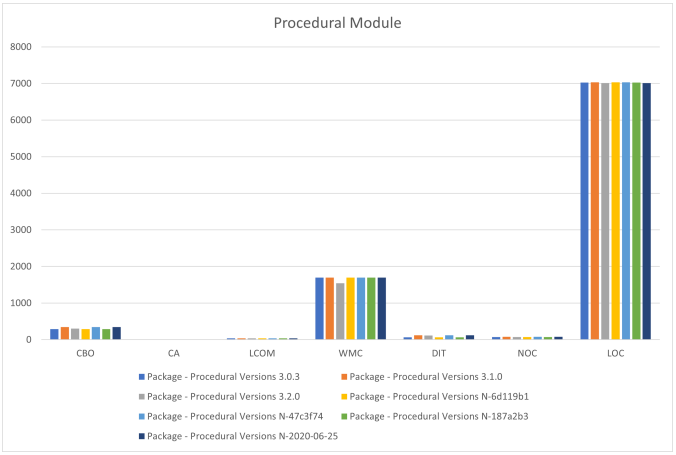

Fig. 16. Procedural Module metrics graph

Package - UI								
	Versions							
Metrics	3.0.3	3.1.0	3.2.0	N-6d119b1	N-47c3f74	N-187a2b3	N-2020-06-25	Avg of all
CBO	133	139	55	137	139	137	141	125.8571429
CA	161	165	146	164	165	163	165	161.2857143
LCOM	13.887	13.92	10.557	13.92	13.92	13.92	13.945	13.43842857
WMC	714	724	440	721	724	721	760	686.2857143
DIT	68	68	10	68	68	68	68	59.71428571
NOC	52	52	52	52	52	52	52	52
LOC	3498	3541	3675	3521	3540	3511	3674	3565.714286

Fig. 17. UI Module metrics tabular form

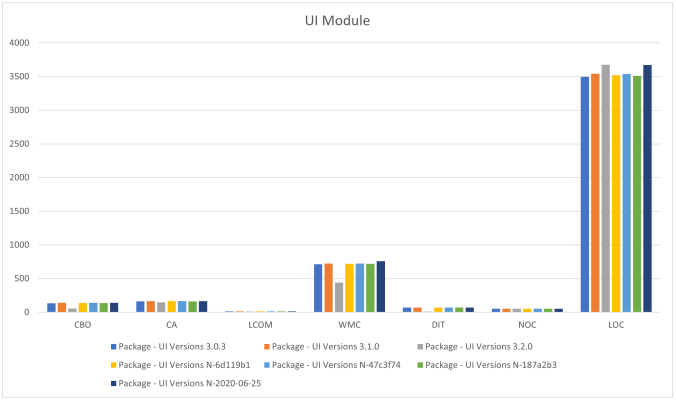

Fig. 18. UI Module metrics graph

Package - Script								
	Versions							
Metrics	3.0.3	3.1.0	3.2.0	N-6d119b1	N-47c3f74	N-187a2b3	N-2020-06-25	Avg of all
CBO	58	58	31	58	58	58	58	54.14285714
CA	3	3	2	3	3	3	3	2.857142857
LCOM	2.306	2.306	1.473	2.306	2.306	2.306	2.306	2.187
WMC	164	164	152	164	164	164	164	162.2857143
DIT	19	19	5	19	19	19	19	17
NOC	15	15	15	15	15	15	15	15
LOC	956	956	962	956	956	956	956	956.8571429

Fig. 19. Script Module metrics tabular form

Fig. 20. Script Module metrics graph

Package - Texture								
	Versions							
Metrics	3.0.3	3.1.0	3.2.0	N-6d119b1	N-47c3f74	N-187a2b3	N-2020-06-25	Avg of all
CBO	370	373	298	371	373	371	373	361.2857143
CA	3	3	2	3	3	3	3	2.857142857
LCOM	14.544	14.544	15.89	14.554	14.544	14.544	14.544	14.73771429
WMC	1040	1041	962	1040	1041	1040	1041	1029.285714
DIT	48	48	40	48	48	48	48	46.85714286
NOC	42	42	42	42	42	42	42	42
LOC	5067	5069	5069	5068	5069	5068	5069	5068.428571

Fig. 21. Texture Module metrics tabular form

having issues in more that on metrics values. The other module is Texture module this is the only module with one metrics value that is not satisfying the maintainability standard and this is considered as less important among all the difficult to maintain modules.

Other modules like Image.filter, Keystroke, Util, Unwrap, animation, and math are also not following maintainability standard in less important metric values such as understandability, size. This is considered as less important because we have observed from the analysis that the evaluation of ArtofIllusion over all the releases has less major changes that affect that maintainability of module in the project. The code structure over all the releases is also observed similar to each other at class level and method level and this is identified from our analysis on WMC, DIT values of the modules and the mean is nearly same to each other.

V. Discussion

Furthermore, from the results we can say that the internal attributes are dependent or connected to each other for example, if the size of the code is high then the complexity of the system is also high which again tells us that the modules are highly coupled and loosely cohesive which is said to be a bad quality software product and it is not easy maintain. ArtOfIllusion does not have a bad structure but there are no major changes evolution of ArtOfIllusion this can be observed from the obtained results since there is no drastic change of selected metric values to evaluated all the seven versions of ArtOfIllusion.

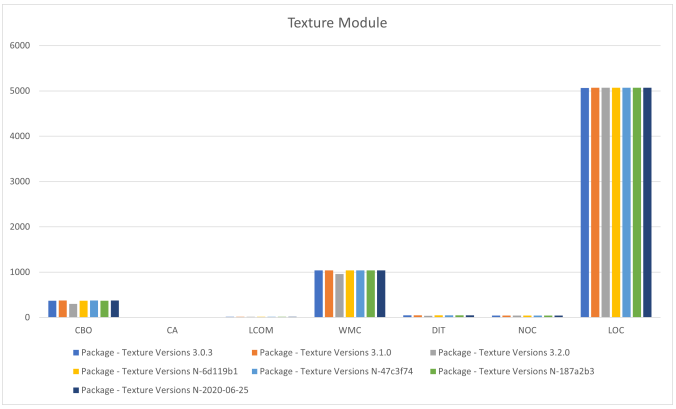

Fig. 22. Texture Module metrics graph

Previously ArtOfIllusion has been used in different empirical studies to evaluate complexity of design in imaging software [12], predicting and assessing the change prone classes [13], evaluating the lack of cohesion metric [3]and a study based on relationship between bug reports and source code [15]. In [12] the authors have designed a study for evaluating the difference of complexity between an imaging software and non imaging software using the CK metrics and some extended complexity metrics. The authors have expected to have a clear distinction in values of complexity between an imaging software and a non imaging software but the results show the opposite. There was not much difference in the complexities of both types of softwares. Malhotra and Ruchika [13] have studied about the prediction and evaluation of change prone classes. They have used two java systems one being ArtOfIllusion and the other system is also a similar product named sweet-home 3D.They have analysed two stable versions of both the systems using different metrics like CBO, NPRM, SLOC,etc.,. from the results they said that NPRM, SLOC and WMC are good change predictors.AL Dallal [3] has performed an empirical study on LCOM and transitive LCOM by applying on two java systems. TLCOM can detect faulty classes more accurately than normal LCOM. Another study where ArtOfIllusion is used for their study is [15] where they explore about the vocabularies of bug reports and the source code.

All the studies have studied ArtOfIllusion in a different way. ArtOfIllusion is a large software system with huge number of classes and lines of code. It is a complex system and difficult to maintain. ArtOfIllusion has been used in the above studies because, though it is complex it does not have much difference from version to version. There is steady growth in each version. All the above studies have used ArtOfIllusion as a system on which they performed different calculations to obtain the results for a particular goal. It was mostly used in comparison studies where it was compared with other similar products. In this project, we have performed a GQM analysis to find out the modules that are difficult to maintain.

The main goal of our project is to evaluate maintainability of ArtOfIllusion. We have collected the data for all the internal attributes given and analyzed based on these internal metrics. Throughout the project it was a very tedious yet satisfying process. We have understood how to measure software, and where can we use these metrics in software processes. We have understood the importance and effectiveness of GQM approach which is the main take away from the course including learning and understanding how to use metric extraction tools like code MR, Understand, and metrics reloaded. Though we have not used all the above mentioned tools in this project but we have used initially to find out which tool works for us.

REFERENCES

[1] Depth of inheritance tree (DIT) - software architect's handbook [book]. ISBN: 9781788624060.

[2] Jehad Al Dallal. Improving the applicability of object-oriented class cohesion metrics. 53(9):914–928.

[3] Jehad Al Dallal. Transitive-based object-oriented lack-of-cohesion metric. 3:1581–1587.

[4] Linda Badri, Mourad Badri, and Fadel Touré. Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems. volume 117, pages 78–92.

[5] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. 20(6):476–493. Conference Name: IEEE Transactions on Software Engineering.

[6] Mike Child, Peter Rosner, and Steve Counsell. A comparison and evaluation of variants in the coupling between objects metric. 151:120–132.

[7] Norman Fenton and James Bieman. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. Taylor & Francis Group, Baton Rouge, UNITED STATES, 2014.

[8] Emanuel Giger, Martin Pinzger, and Harald C. Gall. Can we predict types of code changes? an empirical analysis. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 217–226. ISSN: 2160-1860.

[9] Claire Ingram and Steve Riddle. Linking software design metrics to component change-proneness. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, WETSoM '11, pages 31–37. Association for Computing Machinery.

[10] Ronald Jabangwe, Jürgen Börstler, Darja Šmite, and Claes Wohlin. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. 20(3):640–693.

[11] Lov Kumar, Debendra Kumar Naik, and Santanu Kumar Rath. Validating the effectiveness of object-oriented metrics for predicting maintainability. 57:798–806.

[12] M.E. Larsson and P.A. Laplante. On the complexity of design in imaging software. In *11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'06)*, pages 7 pp.–.

[13] Ruchika Malhotra and Ravi Jangra. Prediction & assessment of change prone classes using statistical & machine learning techniques. 13(4):778–804.

[14] Ruchika Malhotra and Megha Khanna. Mining the impact of object oriented metrics for change prediction using machine learning and search-based techniques. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 228–234.

[15] Laura Moreno, Wathsala Bandara, Sonia Haiduc, and Andrian Marcus. On the relationship between the vocabulary of bug reports and source code. In *2013 IEEE International Conference on Software Maintenance*, pages 452–455. ISSN: 1063-6773.

[16] Alberto S. Nuñez-Varela, Héctor G. Pérez-Gonzalez, Francisco E. Martínez-Perez, and Carlos Soubervielle-Montalvo. Source code metrics: A systematic mapping study. 128:164–197.

[17] Daniele Romano and Martin Pinzger. Using source code metrics to predict change-prone java interfaces. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 303–312. ISSN: 1063-6773.

[18] Frederick T. Sheldon, Kshamta Jerath, and Hong Chung. Metrics for maintainability of class inheritance hierarchies. 14(3):147–160.

VI. ANNEXURES

We have provided tables and graphs related to modules for extra information and clear understanding.

3.0.3

QualifiedName	Name	Complexity	Coupling	Size	Lack of Cohesion
ArtOfIllusion-3.0.3					
<Package>artofillusion	artofillusion	very-high	high	very-high	low
<Package>artofillusion.animation	artofillusion.animation	very-high	high	very-high	low
<Package>artofillusion.animation.distortior	artofillusion.animation.distortion	low-medium	medium-high	medium-high	low
<Package>artofillusion.image	artofillusion.image	low-medium	low-medium	medium-high	low
<Package>artofillusion.image.filter	artofillusion.image.filter	low-medium	low-medium	medium-high	low
<Package>artofillusion.keystroke	artofillusion.keystroke	low	low	low-medium	low
<Package>artofillusion.material	artofillusion.material	low	low-medium	low-medium	low
<Package>artofillusion.math	artofillusion.math	low-medium	low	medium-high	low
<Package>artofillusion.object	artofillusion.object	very-high	high	very-high	low
<Package>artofillusion.procedural	artofillusion.procedural	high	high	very-high	low
<Package>artofillusion.script	artofillusion.script	low	medium-high	medium-high	low
<Package>artofillusion.texture	artofillusion.texture	high	high	very-high	low
<Package>artofillusion.ui	artofillusion.ui	medium-high	high	very-high	low
<Package>artofillusion.unwrap	artofillusion.unwrap	low	low	low-medium	low
<Package>artofillusion.util	artofillusion.util	low	low	low-medium	low
<Package>artofillusion.view	artofillusion.view	low-medium	medium-high	medium-high	low

Fig. 23. Version 3.0.3

3.1.0

QualifiedName	Name	Complexity	Coupling	Size	Lack of Cohesion
ArtOfIllusion-3.1.0					
<Package>artofillusion	artofillusion	very-high	high	very-high	low
<Package>artofillusion.animation	artofillusion.animation	very-high	high	very-high	low
<Package>artofillusion.animation.distortior	artofillusion.animation.distortion	low-medium	medium-high	medium-high	low
<Package>artofillusion.image	artofillusion.image	low-medium	low-medium	medium-high	low
<Package>artofillusion.image.filter	artofillusion.image.filter	low-medium	low-medium	medium-high	low
<Package>artofillusion.keystroke	artofillusion.keystroke	low	low	low-medium	low
<Package>artofillusion.material	artofillusion.material	low	low-medium	low-medium	low
<Package>artofillusion.math	artofillusion.math	low-medium	low	medium-high	low
<Package>artofillusion.object	artofillusion.object	very-high	high	very-high	low
<Package>artofillusion.procedural	artofillusion.procedural	high	high	very-high	low
<Package>artofillusion.script	artofillusion.script	low	medium-high	medium-high	low
<Package>artofillusion.texture	artofillusion.texture	high	high	very-high	low
<Package>artofillusion.ui	artofillusion.ui	medium-high	high	very-high	low
<Package>artofillusion.unwrap	artofillusion.unwrap	low	low	low-medium	low
<Package>artofillusion.util	artofillusion.util	low	low	low-medium	low
<Package>artofillusion.view	artofillusion.view	low-medium	medium-high	medium-high	low

Fig. 24. Version 3.1.0

3.2.0

QualifiedName	Name	Complexity	Coupling	Size	Lack of Cohesion
artofillusion					
<Package>artofillusion	artofillusion	very-high	high	very-high	low
<Package>artofillusion.animation	artofillusion.animation	high	high	very-high	low
<Package>artofillusion.animation.distortior	artofillusion.animation.distortion	low-medium	medium-high	medium-high	low
<Package>artofillusion.image	artofillusion.image	low-medium	low-medium	medium-high	low
<Package>artofillusion.image.filter	artofillusion.image.filter	low-medium	low-medium	medium-high	low
<Package>artofillusion.keystroke	artofillusion.keystroke	low	low	low-medium	low
<Package>artofillusion.material	artofillusion.material	low	low-medium	low-medium	low
<Package>artofillusion.math	artofillusion.math	low-medium	low	medium-high	low
<Package>artofillusion.object	artofillusion.object	very-high	high	very-high	low
<Package>artofillusion.procedural	artofillusion.procedural	high	high	very-high	low
<Package>artofillusion.script	artofillusion.script	low	low-medium	medium-high	low
<Package>artofillusion.texture	artofillusion.texture	medium-high	high	very-high	low
<Package>artofillusion.ui	artofillusion.ui	low-medium	medium-high	very-high	low
<Package>artofillusion.unwrap	artofillusion.unwrap	low	low	low-medium	low
<Package>artofillusion.util	artofillusion.util	low	low	low-medium	low
<Package>artofillusion.view	artofillusion.view	low-medium	medium-high	medium-high	low

Fig. 25. Version 3.2.0

ArtOfIllusion-Nightly-6d119b1					
QualifiedName	Name	Complexity	Coupling	Size	Lack of Cohesion
<Package>artofillusion	artofillusion	very-high	high	very-high	low
<Package>artofillusion.animation	artofillusion.animation	very-high	high	very-high	low
<Package>artofillusion.animation.distortior	artofillusion.animation.distortion	low-medium	medium-high	medium-high	low
<Package>artofillusion.image	artofillusion.image	low-medium	low-medium	medium-high	low
<Package>artofillusion.image.filter	artofillusion.image.filter	low-medium	low-medium	medium-high	low
<Package>artofillusion.keystroke	artofillusion.keystroke	low	low	low-medium	low
<Package>artofillusion.material	artofillusion.material	low	low-medium	low-medium	low
<Package>artofillusion.math	artofillusion.math	low-medium	low	medium-high	low
<Package>artofillusion.object	artofillusion.object	very-high	high	very-high	low
<Package>artofillusion.procedural	artofillusion.procedural	high	high	very-high	low
<Package>artofillusion.script	artofillusion.script	low	medium-high	medium-high	low
<Package>artofillusion.texture	artofillusion.texture	high	high	very-high	low
<Package>artofillusion.ui	artofillusion.ui	medium-high	high	very-high	low
<Package>artofillusion.unwrap	artofillusion.unwrap	low	low	low-medium	low
<Package>artofillusion.util	artofillusion.util	low	low	low-medium	low
<Package>artofillusion.view	artofillusion.view	low-medium	medium-high	medium-high	low

Fig. 26. Version Nightly-6d119b1

ArtOfIllusion-Nightly-47c3f74					
QualifiedName	Name	Complexity	Coupling	Size	Lack of Cohesion
ArtOfIllusion-Nightly-47c3f74					
<Package>artofillusion	artofillusion	very-high	high	very-high	low
<Package>artofillusion.animation	artofillusion.animation	very-high	high	very-high	low
<Package>artofillusion.animation.distortior	artofillusion.animation.distortion	low-medium	medium-high	medium-high	low
<Package>artofillusion.image	artofillusion.image	low-medium	low-medium	medium-high	low
<Package>artofillusion.image.filter	artofillusion.image.filter	low-medium	low-medium	medium-high	low
<Package>artofillusion.keystroke	artofillusion.keystroke	low	low	low-medium	low
<Package>artofillusion.material	artofillusion.material	low	low-medium	low-medium	low
<Package>artofillusion.math	artofillusion.math	low-medium	low	medium-high	low
<Package>artofillusion.object	artofillusion.object	very-high	high	very-high	low
<Package>artofillusion.procedural	artofillusion.procedural	high	high	very-high	low
<Package>artofillusion.script	artofillusion.script	low	medium-high	medium-high	low
<Package>artofillusion.texture	artofillusion.texture	high	high	very-high	low
<Package>artofillusion.ui	artofillusion.ui	medium-high	high	very-high	low
<Package>artofillusion.unwrap	artofillusion.unwrap	low	low	low-medium	low
<Package>artofillusion.util	artofillusion.util	low	low	low-medium	low
<Package>artofillusion.view	artofillusion.view	low-medium	medium-high	medium-high	low

Fig. 27. Version Nightly-47c3f74

ArtOfIllusion-Nightly-187a2b3					
QualifiedName	Name	Complexity	Coupling	Size	Lack of Cohesion
ArtOfIllusion-Nightly-187a2b3					
<Package>artofillusion	artofillusion	very-high	high	very-high	low
<Package>artofillusion.animation	artofillusion.animation	very-high	high	very-high	low
<Package>artofillusion.animation.distortior	artofillusion.animation.distortion	low-medium	medium-high	medium-high	low
<Package>artofillusion.image	artofillusion.image	low-medium	low-medium	medium-high	low
<Package>artofillusion.image.filter	artofillusion.image.filter	low-medium	low-medium	medium-high	low
<Package>artofillusion.keystroke	artofillusion.keystroke	low	low	low-medium	low
<Package>artofillusion.material	artofillusion.material	low	low-medium	low-medium	low
<Package>artofillusion.math	artofillusion.math	low-medium	low	medium-high	low
<Package>artofillusion.object	artofillusion.object	very-high	high	very-high	low
<Package>artofillusion.procedural	artofillusion.procedural	high	high	very-high	low
<Package>artofillusion.script	artofillusion.script	low	medium-high	medium-high	low
<Package>artofillusion.texture	artofillusion.texture	high	high	very-high	low
<Package>artofillusion.ui	artofillusion.ui	medium-high	high	very-high	low
<Package>artofillusion.unwrap	artofillusion.unwrap	low	low	low-medium	low
<Package>artofillusion.util	artofillusion.util	low	low	low-medium	low
<Package>artofillusion.view	artofillusion.view	low-medium	medium-high	medium-high	low

Fig. 28. Version Nightly-187a2b3

ArtOfIllusion-nightly-2020-06-25					
QualifiedName	Name	Complexity	Coupling	Size	Lack of Cohesion
ArtOfIllusion-nightly-2020-06-25					
<Package>artofillusion	artofillusion	very-high	high	very-high	low
<Package>artofillusion.animation	artofillusion.animation	very-high	high	very-high	low
<Package>artofillusion.animation.distortior	artofillusion.animation.distortion	low-medium	medium-high	medium-high	low
<Package>artofillusion.image	artofillusion.image	low-medium	low-medium	medium-high	low
<Package>artofillusion.image.filter	artofillusion.image.filter	low-medium	low-medium	medium-high	low
<Package>artofillusion.keystroke	artofillusion.keystroke	low	low	low-medium	low
<Package>artofillusion.material	artofillusion.material	low	low-medium	low-medium	low
<Package>artofillusion.math	artofillusion.math	low-medium	low	medium-high	low
<Package>artofillusion.object	artofillusion.object	very-high	high	very-high	low
<Package>artofillusion.procedural	artofillusion.procedural	high	high	very-high	low
<Package>artofillusion.script	artofillusion.script	low	medium-high	medium-high	low
<Package>artofillusion.texture	artofillusion.texture	high	high	very-high	low
<Package>artofillusion.ui	artofillusion.ui	medium-high	high	very-high	low
<Package>artofillusion.unwrap	artofillusion.unwrap	low	low	low-medium	low
<Package>artofillusion.util	artofillusion.util	low	low	low-medium	low
<Package>artofillusion.view	artofillusion.view	low-medium	medium-high	medium-high	low

Fig. 29. Version Nightly-2020-06-25

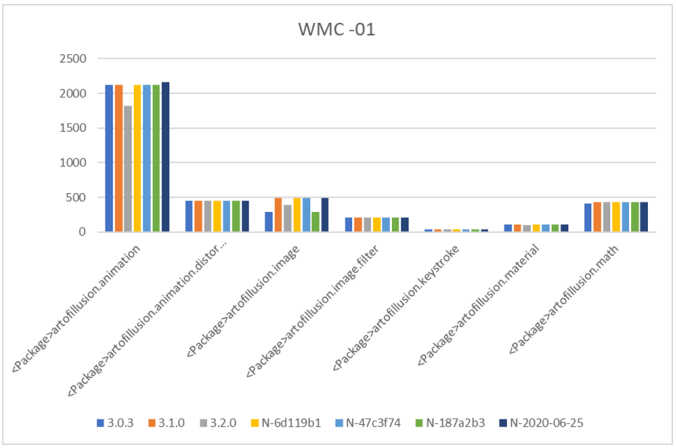

Fig. 30. WMC metrics vs Modules comparison graph1

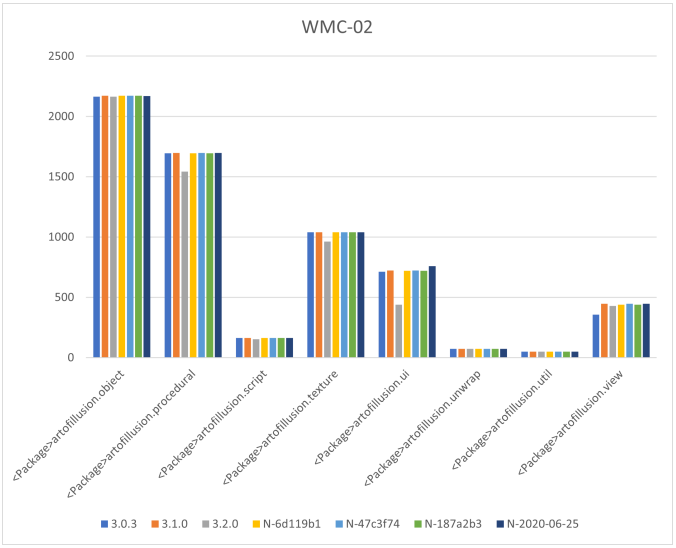

Fig. 31. WMC metrics vs Modules comparison graph2

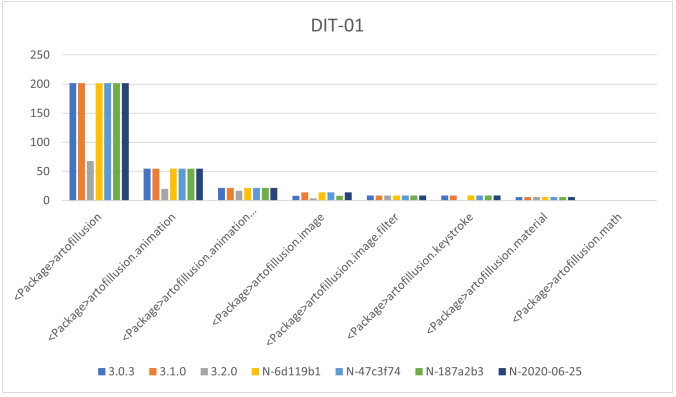

Fig. 32. DIT metrics vs Modules comparison graph1

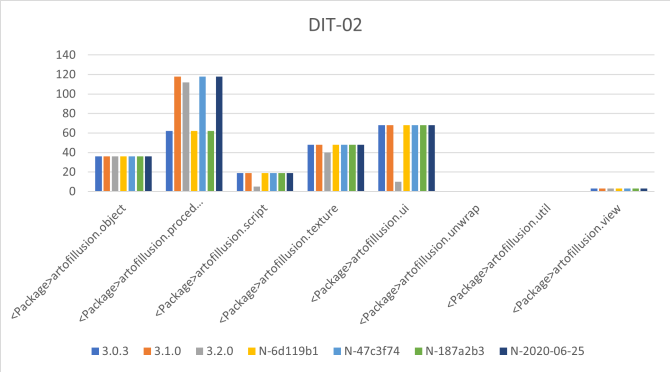

Fig. 33. DIT metrics vs Modules comparison graph2

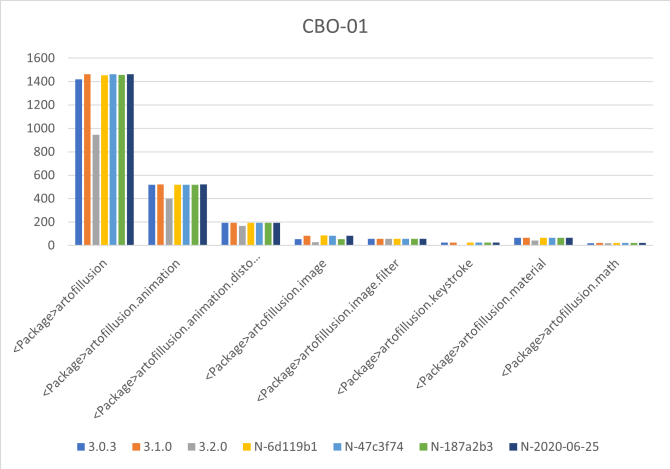

Fig. 34. CBO metrics vs Modules comparison graph1

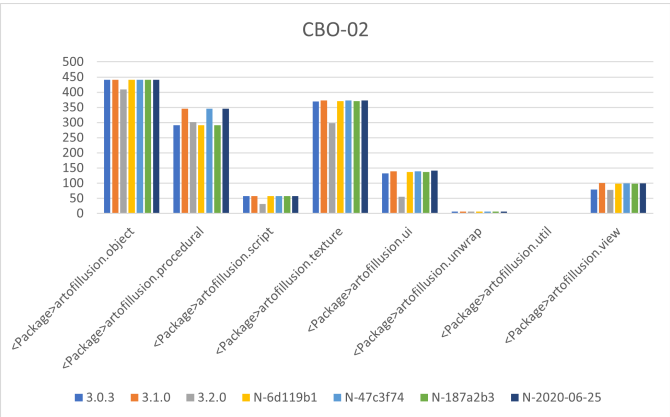

Fig. 35. CBO metrics vs Modules comparison graph2

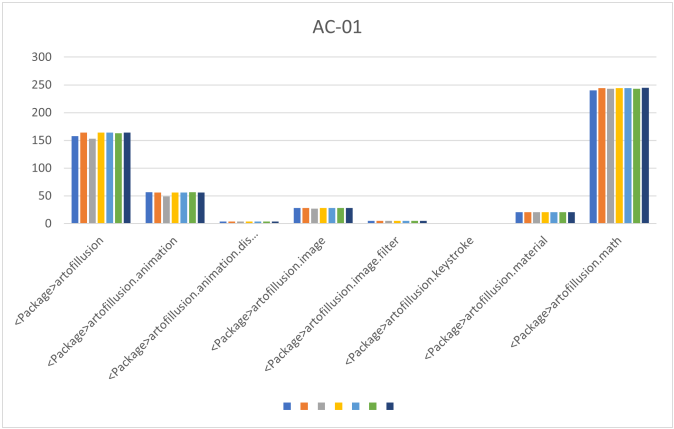

Fig. 36. AC metrics vs Modules comparison graph 1

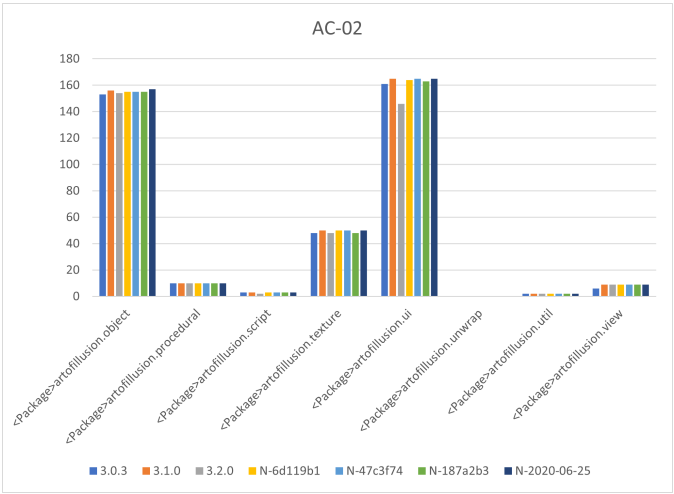

Fig. 37. AC metrics vs Modules comparison graph2

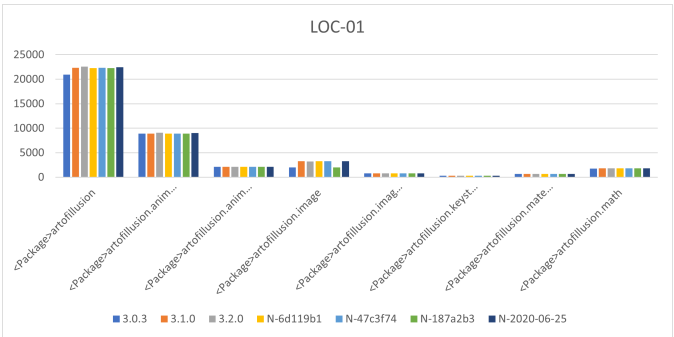

Fig. 38. LOC metrics vs Modules comparison graph1

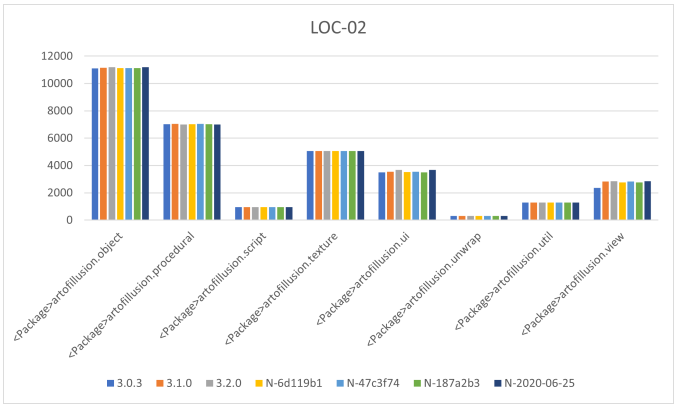

Fig. 39. LOC metrics vs Modules comparison graph2

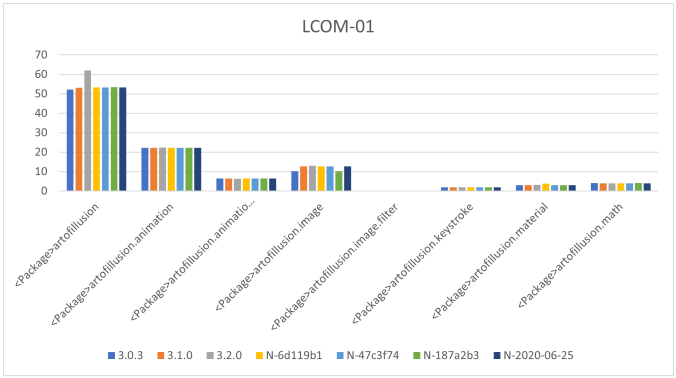

Fig. 40. LCOM metrics vs Modules comparison graph1

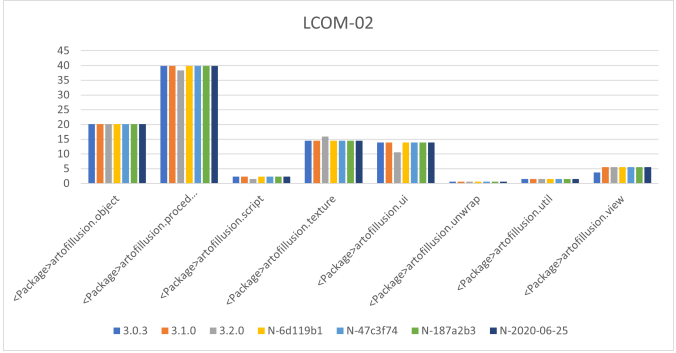

Fig. 41. LCOM metrics vs Modules comparison graph2

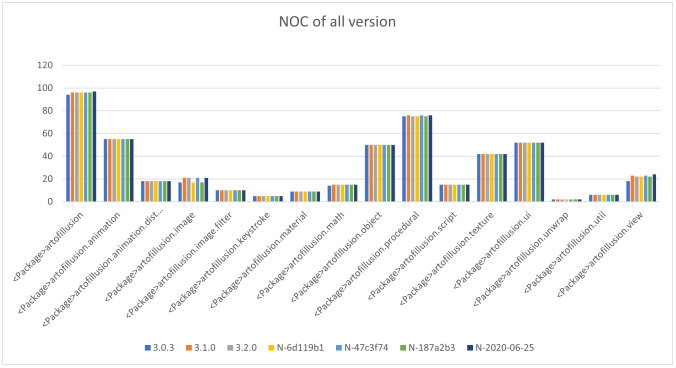

Fig. 42. NOC metrics vs Modules comparison graph