Understandability & Complexity estimation of Software program in Python Language (UCS-PL)

Vishwa Prakash
(SRKUMTSE1804)

M-Tech, Department of Software Engineering, R.K.D.F. Institute of Science & Technology, Bhopal, M.P., India

ABSTRACT

There has been rapid growth in software development. In the Software development process readability, understandability, and maintainability are the important characteristics of software development for quality software, because it may influence the performance of the software. The cost and reuse of the software are also affected by it. Readability and understandability of code/program in python language reduce the testing time of software in the main phase which also, reduces the defects of the software after development. If a developer or a tester can predict the software defects properly then, it reduces the cost, time, and effort as well. Software maintainability measurement early in the development life cycle, especially at the first phase or design phase, may help designers to incorporate other required enhancement/implementation and corrections for improving the performance of the final software. This paper developed to show a comparative analysis of software development in multiple languages like JAVA, C++, and Python. The readability & understandability of the source code depends upon the psychological complexity of the software. A complexity metric for Python language is formulated. Since Python is a high-level OOP language, the present metric is capable to evaluate any object-oriented language. We validate our metric with a case study, comparative study, and empirical validation. The case study is in Python, Java, and C++ and the results prove that Python is better than other object-oriented languages in readability, understandability, and after development maintenance complexity.

Keyword : - *Complexity metric, Python, software complexity, software development, Software Metrics, Object-Oriented Design, UML Class Diagrams, Software Readability, Understandability, and Maintainability, Modifiability etc….*

---

1. INTRODUCTION

Software products are expensive. Therefore, software project managers are always worried about the high cost of software development and are desperately looking for way-outs to cut development costs. A possible way to reduce development costs is to reuse parts from previously developed software. In addition to reduced development cost and time, code reuse also leads to a higher quality of the developed products since the reusable components are ensured to have high level quality. Software metrics determine the degree of maintainability of software application and software products, which is one of the important factors that affect the quality of any kind of software. Even, software metrics provide useful feedback to the developer or designers to impact the decisions that are made during desinign, coding, architecture, or specification phases. Without such feedback, many decisions are made in an ad hoc manner. When programmers try to reuse code, which is written by other programmers, faults may occur due to

misunderstanding of source code. The difficulty of readability and understanding limits the reuse technique. On the Software Development Life Cycle (SDLC) the maintenance phase tends to have a comparatively much longer duration than all the previous phases taken together, obviously resulting in much more effort. It has been reported that the amount of effort spent on the maintenance phase is 65% to 75% of total software development.

In Figure 1, the programmers of the original system were absent, then the other programmers need to reuse the components to enhance the functionalities and correcting faults. Fig.1, shows the communication between programmer developers and software, in the evolution of software systems. Programmer 1 writes the current version of a software system, programmer 2 evolves the next version of that software from the current version. If the written program is difficult to understand, changes to it may cause serious faults, these changes may cost more time than remaking the software systems.

However, That is not easy to measure software understandability because understanding is an internal process of humans.

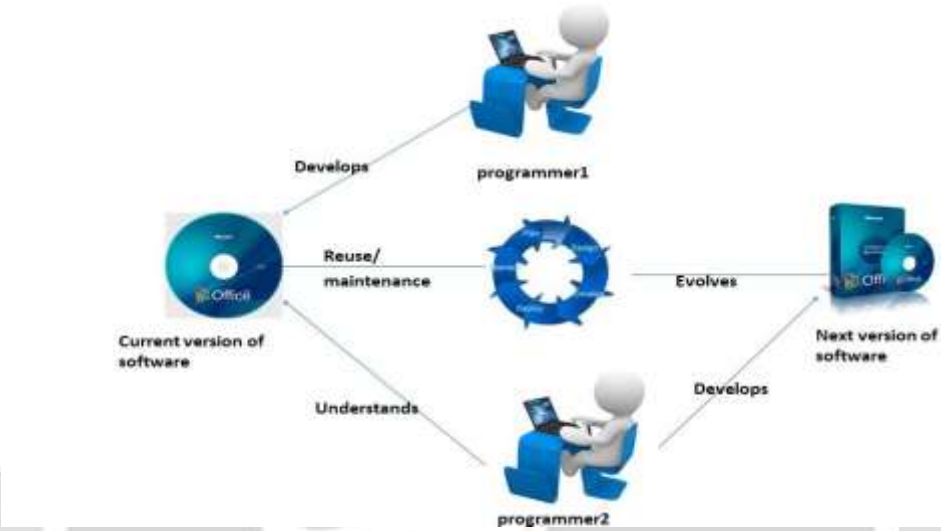

Fig-1 Novice (or New hired) programmer might misunderstand old version and thus introduce faults into new version

1.1 SDLC life-cycle

SDLC stands for Software Development Lifecycle. Its defined as a sequence of development steps followed for developing a software system. The main steps include – planning, analysing, designing, implementation, and maintenance. Testing is also performed on the software product generated like alpha, beta testing. There are various SDLC models the more common being the Waterfall Model & Agile Model. Some the benefit of following the SDLC Lifecycle is the good quality end product such as:

1. Ontology
2. Verification and Validation
3. Software quality
4. Software Maintenance etc...

1.2 Software Development Model

There are different types of models available to implement the Software Development lifecycle (SDLC). The waterfall model is the most common model. This model is like a cascading waterfall. Another model is the Agile Model. This model employs a certain set of values and principles towards software development. In the prototype model, a prototype of the actual product is made for verification. After this model is verified and validated, then only the development of the actual model is started. Other common software models are the Spiral Model, V-Shaped Model, Big Bang Model, the RAD (Rapid Application Development) Model, and the interactive model.

Agile and Waterfall models are common models for software development. The agile model believes that the existing methods need to be tailored to best suit the project requirements and every project needs to be handled differently. In Agile, the tasks are divided into time boxes (small time frames) to deliver specific features for a release. An iterative approach is taken and working software build is delivered after each iteration. In terms of features, each build is incremental, the final build holds all the features required by the customer.

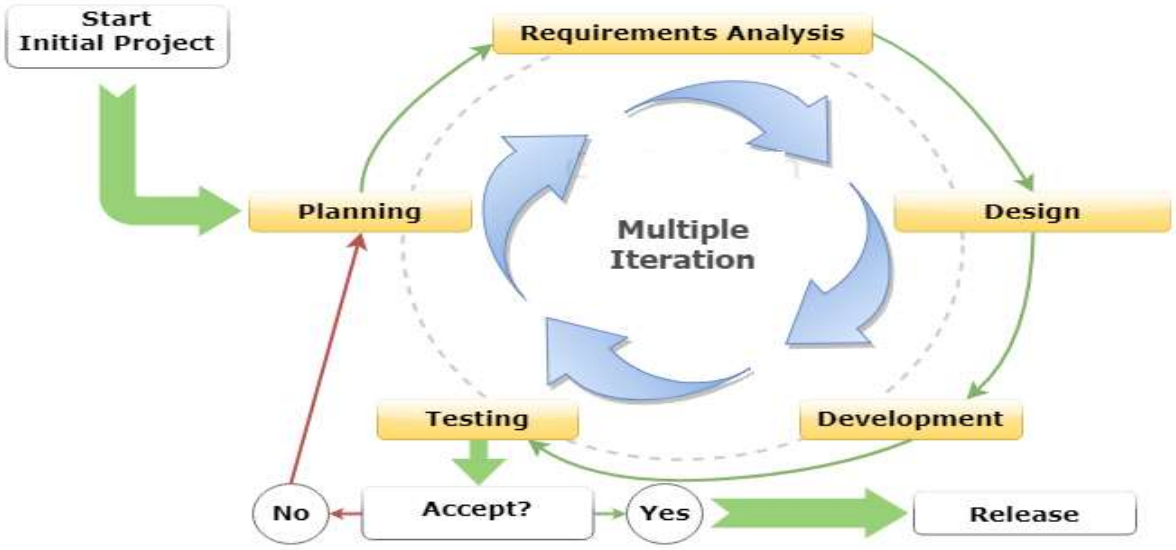

Fig -2: SDLC Agile Software Development methodology

2. SOURCE CODE UNDERSTANDABILITY METRICS

Python is a high-level OOP programming language that lets the programmer work more quickly to integrate systems more effectively. It is free of charge language for commercial purposes. It runs on all major operating systems including Windows, Linux/Unix, Mac OS X, and also has been ported to the Java and .NET virtual machines. Besides those features, Python is an effective language especially for software development for embedded systems. It can also be used in web development.

Understandability of software also requires few metrics. Here a few metrics of code understandability are explained which are used by many organizations. Source code readability quality of documentation should be considered while measuring the software maintainability.

LOC: A common basis of estimate on a software project is the Lines of Code (LOC). Lines of Code is used to create time and cost estimates.

Example: write a code which reads 2 files and writes those files one after the other to the 3rd file.

Code in C	Code in Python

#include<stdio.h>
void main()
{
FILE *f_in,*f_out;
char ch;
f_in=fopen("vp1.txt","r");
f_out=fopen("vp3.txt","w");
ch=getc(f_in);
While(!feof(f_in))
	f_in=open("vp1.txt","r")
	x=f_in.read()
	f_in.close()
	f_in=open("vp2.txt","r")
	y=f_in.read()
	f_in.close()
	s=x+y
	f_out=open("vp3.txt","w")
	f_out.write(s)

```
        {
                putc(ch,f_out);
                ch=getc(f_in);
        }
        fclose(f_in);
        f_in=fopen("vp2.txt","r");
        ch=getc(f_in);
        While(!feof(f_in))
        {
                putc(ch,f_out);
                ch=getc(f_in);
        }
        fclose(f_in)
        fclose(f_out);
}
```

```
f_in=open("vp1.txt","r")
x=f_in.read()
f_in.close()
f_in=open("vp2.txt","r")
y=f_in.read()
f_in.close()
s=x+y
f_out=open("vp3.txt","w")
f_out.write(s)
```

Chart -1 : Code comparison between c and Python**.**

There are used some metrics to calculate understability of software.
1. Comment percent: RSM (Resource Standard Metrics)
2. Function Metrics
3. Function Count Metric
4. Macro Metrics
5. Class Metrics

2.1 Object-Oriented Metrics

The metrics for object-oriented systems focus on measurements that are applied to the class and the design characteristics, for example, encapsulation, information hiding, inheritances, localization, etc. So Object-oriented metrics are usually used to assess the quality of software designs and development.

Very few metrics have been proposed for OOP software systems.

The Chidamber and Kemerer Metrics (C&K) Suite.
The Chidamber and Kemerer Metrics (C&K) suite includes the following metrics:
1. Weighted methods per class (WMC):
2. Depth of Inheritance Tree (DIT):
3. Number of Children (NOC):
4. Lack of Cohesion in Methods (LCOM):
5. Coupling Between Objects (CBO):
6. Response For a Class (RFC):

The Lorenz and Kidd Metrics suite
Unlike C & K metrics the most of the L & K metrics are directly measures are include directly countable measures. Those metrics are:
1. Number of Public Methods:
2. Number of Methods:

3. Number of Public variables per class:
4. Number of Variables per Class:
5. Number of Methods Inherited by Subclass:
6. Number of Methods Overridden by subclass:
7. Number of Methods added by Subclass:
8. Average Method Size:
9. Number of Times a class is reused:
10. Number of Friends of a class:

Abreu Metrics

The emphasis behind the development of the metrics is on the features of encapsulation, inheritance, and coupling. The six Abreu Metrics can be summarized as:

1. Polymorphism Factor:
2. Coupling Factor:
3. Method hiding factor:
4. Attribute Hiding Factor:
5. Method Inheritance Factor:
6. Attribute Inheritance Factor:

3. PROPOSED METRIC

If we analyze object-oriented software, we will find that software consists of several classes with the main program. Accordingly, the complexity of the object-oriented code depends on the complexity of the class and the complexity of the main program. It is a common observation that most of the available metrics do not care for the complexity of the main program in object-oriented systems. In our proposal, we consider all the factors, which are responsible for increasing the complexity of the class as well as the main program. Actually, the main program is the main component, which differentiates Python with other object-oriented languages. Further, the complexity of a system depends on the following factors:

Complexity due to classes: Class is a basic unit or part of object-oriented software development. All the functions are distributed in different classes. Further classes in the object-oriented code either are in inheritance hierarchy or distinctly distributed. Accordingly, the complexity of all the classes is due to classes in the inheritance hierarchy and the complexity of distinct classes.

Complexity due to global factors: The second important factor, which is normally neglected in calculating the complexity of object-oriented codes, is the complexity of global factors in the main program.

Complexity due to coupling: Coupling is one of the most important factors for the increasing complexity of object-oriented code. The other factors like cohesion and methods are considered the complexity factors inside the class. Accordingly, we propose that the Complexity of the Python code is defined as:

Software Metric for Python (SMPy) = CDclass+ Cglobal + CIclass + Ccoupling (1)

CIclass = Complexity due to Inheritance
CDclass = Complexity of Distinct Class
Cglobal= Global Complexity
Ccoupling= Complexity due to coupling between classes.

Table -1: Values of structures

Category	Value	Flow Graph
Sequence	1	
Condition	2	
Loop	3	
Nested Loop	3	-
Function	2	
Recursion	3	
Exception	2	

4. EMPIRICAL VALIDATION

The empirical validation of the metric is carried out through a case study and comparative study, and application of metric on a real project. The case study under consideration is taken into three different languages. We compare our metric with the most popular CK metrics suite. The case study, comparative study, and observations on real projects are demonstrated in sections 4.1, 4.2, and 4.3 respectively.

4.1 Case study

we chose a case study for the practical applicability of our metric. We measure a code, which covers cohesion, inheritance, polymorphism, coupling, attributes, methods, etc. This code, which covered most of possible coding features, is tried in three different languages: C++, Java, and Python. Its UML figure is given in Fig 3. In other words, we try to develop a system in three different languages and then estimate the complexity of the same system in three different languages. we considered the same system for the demonstration of metric. We have already calculated the complexity of the system in Python language in the same section. Now we are estimating the complexity of the same example in C++ and Java. Their class complexity and non-Class complexity are given such as:

The metric values for C++ :
Cclass(Colour)=21
Cclass(Shapes)=7
Cclass(Figure1P)=7
Cclass(Square)=29
Cclass(Circle)=29
Cclass(Figure2P)=11
Cclass(Rectangle)=29
Cclass(Oval)=29

Table -2: Non-class complexity

Non-Class	var+str+obj	Complexity
Cglobal	32	32

CIclass=Shapes*(Figure1P*(Square+Circle+Figure2P*(Rectangle+Oval))) (2)
=7*(7*(29+29+11*(29+29))) = 34104
CDclass=21
Cglobal=32
Total complexity of the system in
C++=CIclass+CDclass+Cglobal+coupling (3)
=34104+21+32+2 = 34173

The metric Values for the Java:
Cclass(Colour)=21
Cclass(Shapes)=7
Cclass(Figure1P)=7
Cclass(Square)=29
Cclass(Circle)=29
Cclass(Figure2P)=11
Cclass(Rectangle)=29
Cclass(Oval)=29

Table -3: Non-class complexity

Non-Class	var+str+obj	Complexity
Cglobal	71	71

CIclass=Shapes*(Figure1P*(Square+Circle+Figure2P*(Rectangle+Oval))) (4)
=7*(7*(29+29+11*(29+29))) = 34104
CDclass=21
Cglobal=71
Total complexity of the system in JAVA= CIclass + CDclass + Cglobal + coupling (5)
=34104+21+71+2 = 34212

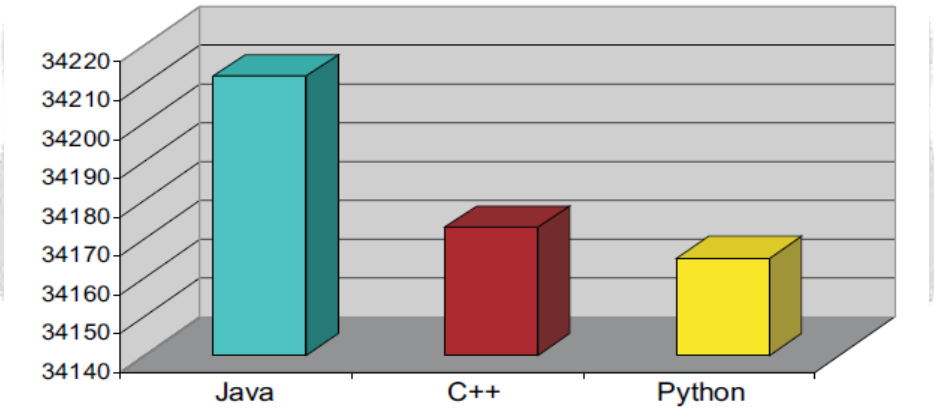

Fig -3: Comparison between languages Complexity

The data obtained from the above experimentations provide very valuable information regarding the metrics as well as the features of the languages. We can very easily observe that the metric values for classes in all three languages are the same and are: 21, 7, 7, 29, 29, 11, 29, and 29 for classes Colour, Shapes, Square, Circle, Figure1P, Figure2P, Rectangle, and Oval.
However the complexity for the whole system in Python, C++ and Java is different and is 34165, 34173 and 34212 respectively (Fig. 3). Here it is important to note that at the class level the complexity values are equal and the differences occur at the main program. This is the reason why the complexity of the same project is different in three different languages. The complexity of the whole system is the least in python language. It proves the uniqueness of the Python language.

4.2 Comparison with the other metrics

We compare the proposed metric with a well-known CK metric suite. All the metrics are applied to the classes of the case study under consideration. The metric values for different classes are summarised in Tab. 4. We found that metric values for SMpy are higher than all the CK measures as we compared our metric with CK metric suites, . It is because of the fact that SMpy includes all the parameters, which are responsible for the complexity of the systems.

However, all parameters were calculated individually in the CK metric suites. In other words, we can say that SMpy is the superset of all the measures proposed by Chidamber et al.

Table -4: Non-class complexity

Metric \ Class	Shapes	Figure1P	Square	Circle	Figure2P	Rectangle	Oval	Colour
WMC(1)	3	2	2	2	2	2	2	2
RFC	3	5	7	7	7	9	9	2
DIT	0	1	2	2	2	3	3	0
NOC	1	3	0	0	2	0	0	0
LCOM	2	2	0	0	2	0	0	0
CBO	0	1	1	1	1	1	1	0
SMpy	7	7	29	29	11	29	29	21

4.3 **A real project**

The practical usefulness of a new measure cannot be proved without the proper empirical validation which includes the applicability of the metric on real projects. For proper empirical validation purposes, we select an open-source project available on the Web. We believe that the open-source code is more beneficial for the readers because they can also evaluate the project in the same way as the original author does. The selected project is a cross-platform set of Python modules designed for writing computer-based video games. It includes libraries for computer graphics & sound and designed to be used with the Python programming language. This is built over the Simple Direct Media Layer (SDML) library, with the intention of allowing real-time computer game development without the low-level mechanics of the C programming language and its derivatives. This is based on the assumption that the most expensive functions inside games (mainly the graphics part) can be completely abstracted from the game logic in itself, to structure the game, making it possible to use a high-level programming language like Python. We estimate the complexity of each class independently. Classes are coupled in two ways: through inheritances and message calls. The inheritance hierarchies of the classes coupled through the inheritance are shown in Fig. 4. Some classes are independent and therefore not affected due to inheritance and are shown in Fig. 5.

Table -5: Class complexities

Class	Attributes	Structures	Variables	Objects	MA	AM	Cohesion	Cclass
GameObject	1	1	0	0	1	1	1	3
MapObject	7	27	12	0	1	6	0.1	46.1
Level	2	18	10	0	2	2	1	31
Level_zero	1	1	0	0	1	2	0.5	2.5
Level_one	3	52	4	1	1	2	0.5	60.5
ImagedObject	2	6	0	0	0	0	0	8
PropTile	0	1	0	0	0	0	0	1
ActorTile	2	3	1	1	0	0	0	8
Sphere	9	9	7	1	2	5	0.4	26.4
Background	1	2	2	0	0	0	0	5
FloorTime	0	1	0	0	0	0	0	1
Cement	3	2	0	0	0	0	0	5
Grass	3	8	1	0	1	1	1	13
Curb	3	2	0	0	1	1	1	6
Void	3	8	3	0	1	1	1	15
Widget	3	24	5	0	4	6	0.6	32.6
Button	10	32	6	0	6	9	1	48.6
DirectionButtons	2	15	4	0	2	2	0.6	22
ViewPort	5	9	0	0	3	5	0.6	14.6
GameObjects	3	7	0	0	3	3	1	11

Clock	11	24	4	0	4	7	0.5	39.5
State	0	1	0	0	0	0	0	1
CyclePath	8	50	6	0	4	8	0.5	64.5

In Tab. 5, the complexity of each class is shown. In the first column the name of the class is given. For different parameters, the metric values which affect the complexity of the class i.e. attributes, variables, structures, objects and cohesion are given in column 2 – 7. Cclass is calculated by the equations.
It is very easy to observe that the complexity of the class highly depends on its parameter. The highest Cclass is 48.6 for Button class, which due to complex structure of its methods number of attributes, and number of variable. The lowest values are for those classes that have simplest structure, for i.e the class State, Floortile and Proptile. The non-classes complexity defined as Cglobal is due to global variables structures and objects and is computed in Tab. 6. It can be easily observed that global complexity also plays an important role in increasing the overall complexity

Table - 6: Class complexities

Non-Class	var+str+obj	Complexity
Cglobal	71	71

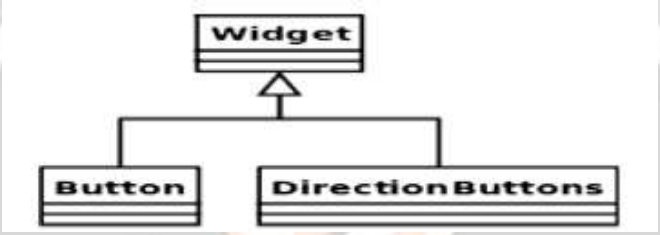

Fig -4: Inheritance 1

There are different class hierarchies in this project. The first-class hierarchy is shown in Fig. 4. In this hierarchy two classes Button and Direction Buttons are on the same level and inherited from class Widget. Due to the effect of this inheritance the complexity of the class Widget is computed as follows:
Widget (Button + DirectionButtons) (6)
=32.6(48.6 + 22)
=2301.5
Another class hierarchy which includes 15 classes is shown in Fig. 5. The class Game object is at the top of the hierarchy. The complexities of each class under inheritance are given in Tab. 7. The complexity due to inheritance is computed as 275591.1. The demonstration of the
calculation for inheritance is given in the following paragraphs.

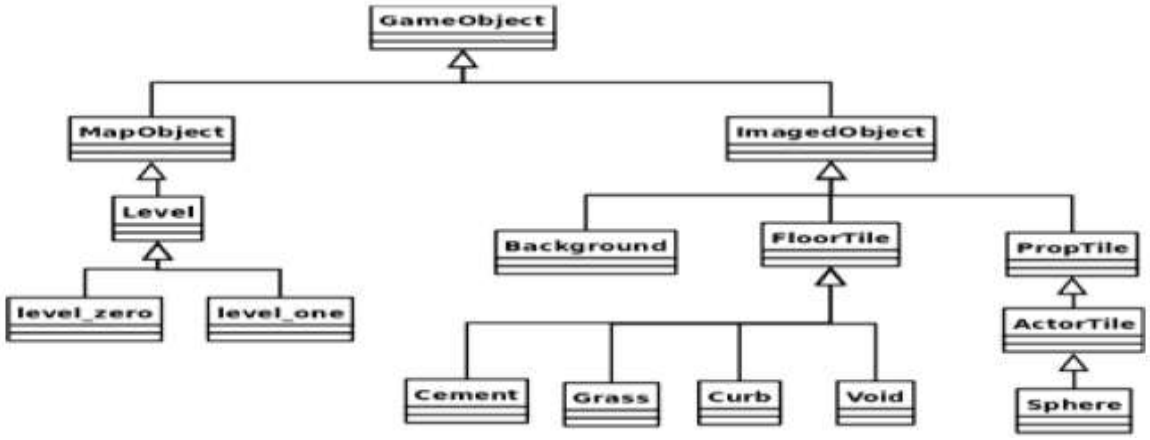

Fig -5: Inheritance 2

GameObject[MapObject(Level(level_zero + level_one)) + ImagedObject(Background + FloorTile(Cement + Grass + Curb + Void) + Proptile(ActorTile(Sphere)))] (7)
=3[46.1(31(2.5+60.5)) + 8(5 + 1(5 + 13 + 6 + 15) + 1(7(26.4)))]
=275591.1

The different complexities for the project are summarised as follows:
CIclass=277892.6
CDclass=130.6
coupling=0
Cglobal=1304

Table - 6: Class complexities

Class	Complexity
Widget	32.6
Button	48.6
DirectButton	22
	3
GameObject	3
MapObject	46.1
ImagedObject	8
Level	31
Background	5
FloorTile	1
PropTile	1
Level_zero	2.5
Level_one	60.5
Cement	5
Grass	13
Curb	6
Void	15
ActorTile	7
Sphere	26.4

Based on the different complexity values due to different factors, the overall complexity of this project is computed as:
SMPy= CIclass + CDclass + coupling + Cglobal (8)
SMPy = 279327.2

The above computation proves the applicability of SMPy on real-world applications. This also proves that not only one factor is responsible for the complexity of the whole code but also there are several factors, which plays the important role in increasing the overall complexity of the code. It is worth mentioning that all these factors are not new but up to now, these factors have not been unified for complexity calculation purposes. The complexities for all these factors like inheritance, coupling, methods, are computed independently in the available complexity metric. It is our first attempt to unify all of them in a single metric. In addition, we tried to implement it on the project written in Python. In section 4.1, our experimentation proves that Python is comparatively a better language for the OOP software development.

5. CONCLUSIONS

Software understandability affects the quality of overall software engineering. If software understandability is favorable, the software development process can be mastered definitely. There are many metrics for evaluating the quality of codes written in different languages. However, There is no efforts have been done to propose metrics for Python. which is an important and useful language especially for software development for the embedded systems. In this paper, we are trying to investigate all the factors, which are responsible for increasing the complexity of code written in Python language. Accordingly, we have proposed a unified metric for this Python language. The practical

applicability of the metric is demonstrated in a case study. We have also validated our work with an empirical validation study by applying it to a real-world project. We hope that the present work will attract the attention of the researchers and practitioners who are working in the OOP domain, especially those using Python. As future work, we aim to provide a list of common words (meaningful names) in object-oriented languages and programming language, which can be used for naming the variables and attributes, while writing the code.

6. ACKNOWLEDGEMENT

I owe deep gratitude to the ones who have contributed greatly to completion of this research. At the outset, I would like to express my sincere thanks to **Prof. Dinesh Kumar Sahu** (HOD-SE), for his continuous encouragement, support and advice during our research work. As our supervisor, he has constantly encouraged me to remain focused on achieving my goal. His observation and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. He has helped me greatly and been a source of knowledge. I am also thankful to all professors of the department for their support.

7. REFERENCES

[1] Costagliola, G.; Tortora, G. Class points:An approach for the size Estimation of Object-oriented systems. IEEE Transactions on Software Engineering,
[2] Misra, S.; Akman, I.Weighted Class Complexity:AMeasure of Complexity for Object-Oriented Systems. Journal of Information Science and Engineering,
[3] Chidamber, S. R.; Kermer, C. F. A Metric Suite for object oriented design. IEEE Transactions Software Engineering,
[4] http://www.Python.org/about/success/carmanah/
[5] Lutz, M. Learning Python, 4th Edition, Ebook, Safari Books Online, Publisher: O'Reilly Media
[6] Fenton, N. E.; Pfleeger, S. L. Software Metrics: A Rigorous and Practical Approach, 2nd Edition Revised ed. Boston:
[7] Python Programming Language, Available from: http://www.Python.org/
[8] Misra, S.; Akman, I. Unified Complexity Metric: A measure of Complexity, Proc. of National Academy of Sciences SectionA.
[9] Basci, D.; Misra, S. Measuring and Evaluating a Design Complexity Metric for XML Schema Documents, Journal of Information Science and Engineering,
[10] Wang, Y.; Shao, J. A New Measure of Software Complexity Based On Cognitive Weights. Can. J. Elec. Comput. Engg,
[11] SciPy.in 2019, https://scipy.in/2019
[12] Confoo.Ca Web Techno Conference, Available from: http://www.confoo.ca/en
[13] Neuroscience – Brain vs. Computer Available from: http://faculty.washington.edu/chudler/bvc.html
[14] Computer vs. The Brain, Available from: http://library.thinkquest.org/C001501/the_saga/sim.htm
[15] Python Code Complexity Metrics And Tools available from: http://agiletesting.blogspot.com/2008/03/Python-codecomplexity-metrics-and.html
[16] Measuring Cyclomatic Complexity f Python Code available at:http://www.traceback.org/2008/03/31/measuringcyclomatic- complexity-of-Python-code/
[17] Andersson, M.; Vestergren, P. Object-Oriented Design Quality Metrics, Uppsala Master's Theses in omputer Science 276, ISSN 1100-1836
[18] Misra S. Ferid C.ASoftware Metric for Python Language, Proc. of ICSSA2010, (ICSSA 2020).
[19] http://jtauber.com/pyso/ [35] Dufour, B.; Driesen, K. Hendren, L. Verbrugge, C. Dynamic metrics for java.SIGPLAN Notics, 38, 149-168.
[20] Rsm metrics. Website. http://msquaredtechnologies.com/m2rsm/docs/rsm metrics.
[21] Understandability metrics. Website. http:www.aivosto.com/project/help.

BIOGRAPHY

	I'm from a middle-class family of Bihar. Nobody comes in this world, without the support of family and friends, I always respect them. My mother and father is real god for me. They always blessing their child in every condition. Everybody has an ambition in life. Aim or ambition is the inner aspiration of man. No man can do anything in the world without aim. I always follow these two lines 'DO YOUR WORK AND LITTLE MORE' & 'NEVER GIVE UP'. I am a technofreak, I have a strong interest in Technologies and their architecture developments.