**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Bawana Road, Delhi 110042

**DEPARTMENT OF SOFTWARE ENGINEERING**



SE427: SOFTWARE PROJECT MANAGEMENT LAB

**SUBMITTED TO**

**Ms. Kishwar Khan**

(PhD)

*DEPT. OF SOFTWARE ENGINEERING*

**Mr. Prathmesh**

(M. Tech.)

*DEPT. OF SOFTWARE ENGINEERING*

**Mr. Manoj**

(M. Tech.)

*DEPT. OF SOFTWARE ENGINEERING*

**SUBMITTED BY**

**BHASKAR ANAND**

2K20/CO/121

(B. Tech.)

*DEPT. OF COMPUTER ENGINEERING*

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Aim** | **Date** | **Signature** |
| **1** | Write a program to implement function point analysis method | 18-Aug-23 |  |
| **2** | Write a program to implement Walston- Felix Model and SEL Model and compare both. | 25-Aug-23 |  |
| **3** | Write a program to implement Basic COCOMO | 08-Sept-23 |  |
| **4** | Write a program to implement Intermediate COCOMO | 22-Sept-23 |  |
| **5** | Write a program to implement Detailed COCOMO | 13-Oct-23 |  |
| **6** | Write a program to implement Application Composition Estimation Model for effort estimation | 20-Oct-23 |  |
| **7** | Write a program to implement Early Design Model and calculate the effort for the development of project | 27-Oct-23 |  |

EXPERIMENT 1

**Write a program to implement function point analysis method**

## THEORY:

### Function Point Analysis (FPA) is a robust software estimation technique utilized in the field of software engineering to quantify the functionality offered by a software application. The primary objective of FPA is to assign a numerical value, known as Function Points (FP), to a software system based on the functionalities it provides to the end-users. Function Points are a measure of the logical functionality and business value of a software application, independent of the underlying technology, coding, or implementation details.

### To determine Function Points, FPA considers five fundamental components, each with a specific weight:

### External Inputs (EI): These represent user interactions that result in data being input into the system. An example might be a user entering customer details into a CRM system. EIs are weighted higher as they often require more processing and validation.

### External Outputs (EO): These refer to user interactions that lead to data being displayed or output from the system. An example could be generating a report from a database. EOs are also weighted moderately.

### External Interfaces (EQ): These are functionalities that involve interfacing with external systems or software. For instance, exchanging data with third-party APIs or integrating with other applications. EQs are given a lower weight.

### Internal Logical Files (ILF): ILFs represent logical groups of data within the software that are maintained and controlled by the system. They are weighted higher because they often require extensive data management.

### External Interface Files (EIF): EIFs are similar to ILFs but represent data entities referenced by the software but maintained by other applications. EIFs receive a moderate weight.

### To calculate the total Function Points, the FPA method multiplies the count of each component by its respective weight and sums them. This results in a single numerical value that signifies the overall size and complexity of the software system.

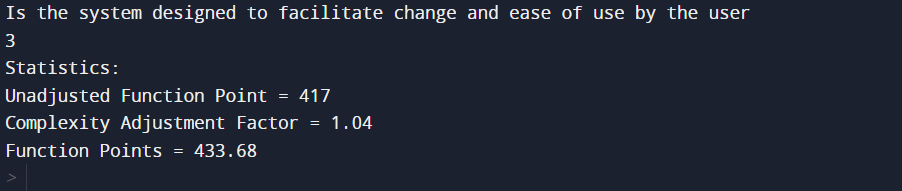
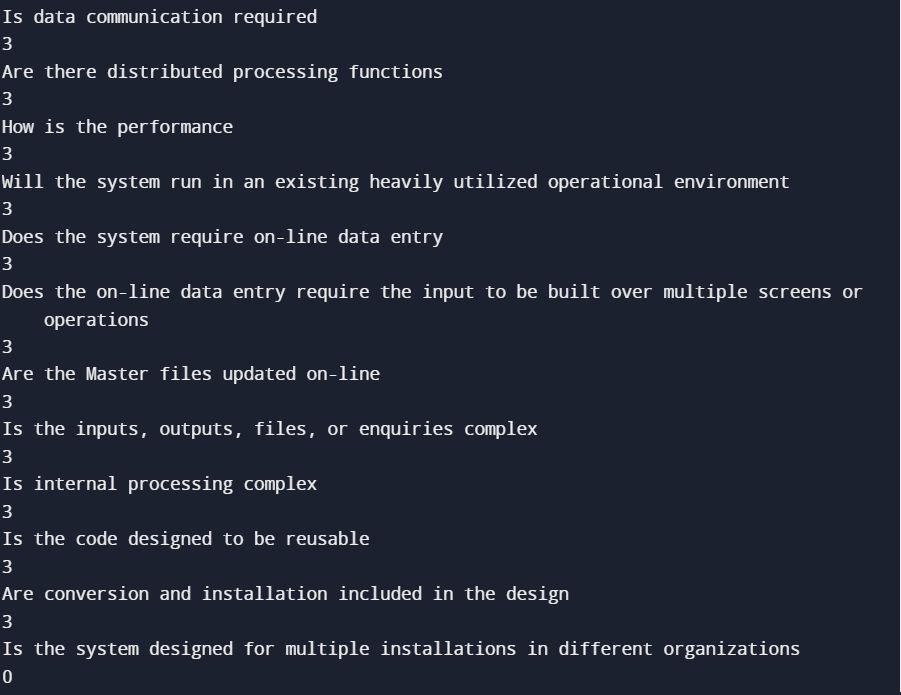
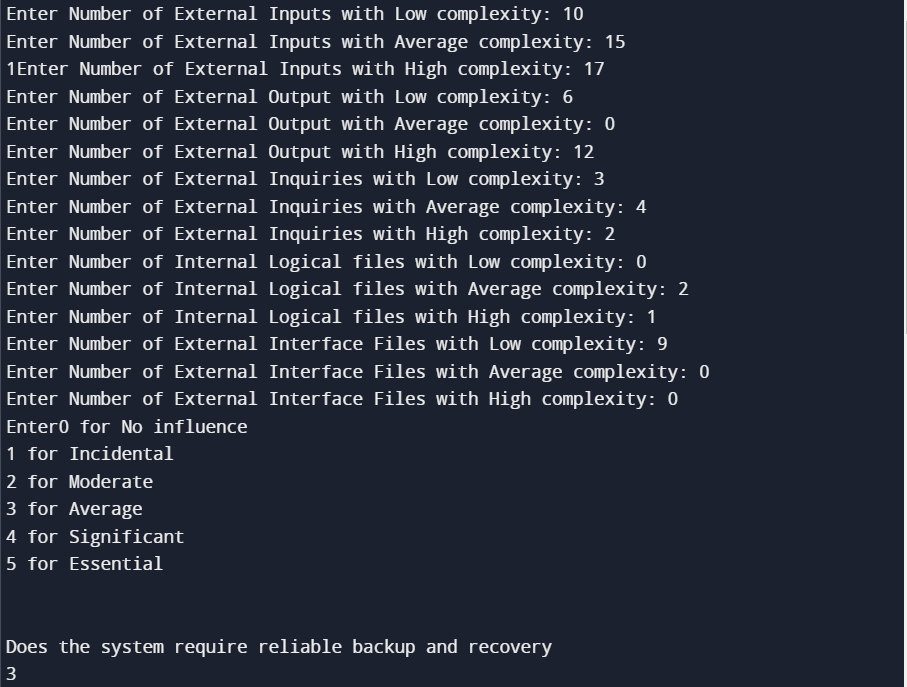
### Function Point Analysis is an invaluable tool for software project estimation, as it enables more accurate planning, resource allocation, and cost estimation. By abstracting the software's functionality from its implementation details, FPA helps in making comparisons across different projects and technologies, facilitating better decision-making in software development and project management.

### CODE:

### 

### 

### OUTPUT OF THE PROGRAM:



### LEARNING OUTCOME:

### By conducting this experiment, we have gained a practical understanding of how to implement a simple Function Point Analysis method in Python.

EXPERIMENT 2

**Write a program to implement** **Walston- Felix Model and SEL Model and compare both.**

## THEORY:

**Walston-Felix Model:**

The Walston-Felix Model is a classic software complexity estimation model that provides insight into the complexity of a software project, particularly focusing on the code itself. The model was developed by Charles R. Walston and Raymond A. Felix. It relies on the principle that as the codebase becomes more complex, it generally requires more effort and resources for development, maintenance, and debugging.

In the Walston-Felix Model, software complexity is assessed based on the number of operators and operands within the code. Operators represent the various operations performed in the code, while operands are the variables, constants, and data manipulated by these operations. The model utilizes a simple formula to calculate complexity, often in terms of man-hours required for development or maintenance.

A higher number of operators and operands in the code results in a higher calculated complexity, suggesting that the software may be more challenging to work with. This model serves as a valuable tool for software project managers and developers to estimate the expected level of effort needed for coding, debugging, and maintaining software.

**Software Engineering Laboratory (SEL) Model:**

The Software Engineering Laboratory (SEL) Model is a comprehensive software complexity estimation model that considers a broader range of factors. Developed by the Software Engineering Laboratory at NASA's Goddard Space Flight Center, this model takes into account aspects beyond just the codebase, such as project size, development environment, and the experience level of the development team.

The SEL Model aims to provide a more holistic view of the software development process. It incorporates elements like the size of the software in KLOC (Kilo Lines of Code), an environment factor that accounts for the tools and infrastructure used, and an experience factor reflecting the expertise of the development team. By considering these additional factors, the SEL Model offers a more comprehensive assessment of software complexity.

The model's results can guide project managers in making informed decisions about resource allocation, project scheduling, and risk assessment. It is particularly valuable for large-scale software projects where various factors can influence the complexity of development and maintenance.

When comparing the Walston-Felix Model and the SEL Model, it's important to recognize that they serve different purposes. The Walston-Felix Model is more code-centric, focusing on the code's intrinsic complexity, while the SEL Model takes a broader approach, considering external factors that may influence software development. Therefore, the choice between these models depends on the specific needs of a project and the level of detail required for complexity estimation.

### CODE:

### 

### OUTPUT OF THE PROGRAM:

### 

### Learnings:

We have successfully learnt the way to implement Walston- Felix Model and SEL Model in python.

EXPERIMENT 3

**Write a program to implement Basic COCOMO**

## THEORY:

COCOMO (Constructive Cost Model) is a regression model based on LOC, i.e. number of Lines of Code. It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the COCOMO are primarily Effort & Schedule:

* **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
* **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

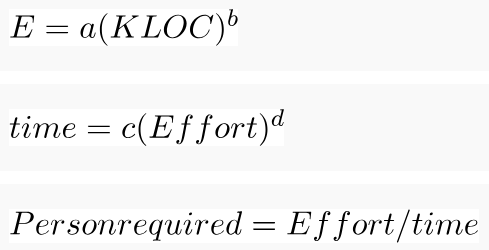
Different models of COCOMO have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of the constant to be used in subsequent calculations.

These characteristics pertaining to different system types are mentioned below. Boehm’s definition of organic, semidetached, and embedded systems:

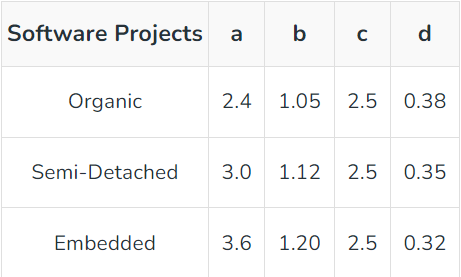
1. **Organic –** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
2. **Semi-detached** – A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.
3. **Embedded** – A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

* **Basic COCOMO Model**
* **Intermediate COCOMO Model**
* **Detailed COCOMO Model**

1. **Basic Model –**



The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a, b, c, and d for the Basic Model for the different categories of the system:



The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. The development time is measured in months. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, and expertise is taken into account, henceforth the estimate is rough.

### CODE:

### 

### OUTPUT OF THE PROGRAM:

### 

## LEARNINGS:

### By conducting this experiment, we have gained a practical understanding of how to implement basic COCOMO model in Python.

EXPERIMENT 4

**Write a program to implement Intermediate COCOMO**

## THEORY:

Intermediate COCOMO, short for "Constructive Cost Model," is an evolution of the original COCOMO model developed by Dr. Barry Boehm. It is a widely used software cost estimation model that provides a more nuanced and accurate prediction of the effort, development time, and personnel requirements for software projects. Intermediate COCOMO is particularly valuable for addressing the complexities and variations present in modern software development.

Key components and concepts of the Intermediate COCOMO model include:

**Lines of Code (KLOC):** One of the fundamental inputs for the model is the estimated size of the software project in thousands of lines of code. The size of the codebase is a critical factor that impacts project effort and cost.

**Development Mode:** Intermediate COCOMO distinguishes between three development modes:

* ***Organic Mode:*** This mode applies to small to medium-sized projects developed by experienced teams with a high level of cooperation. The codebase is relatively straightforward and well-structured.
* ***Semi-Detached Mode:*** Projects in this mode are of medium size and complexity, involving a mix of experienced and moderately experienced team members. The codebase may have some complexities.
* ***Embedded Mode:*** Reserved for large, complex projects that involve a mix of experienced and inexperienced team members. The codebase is often intricate and embedded in other systems.

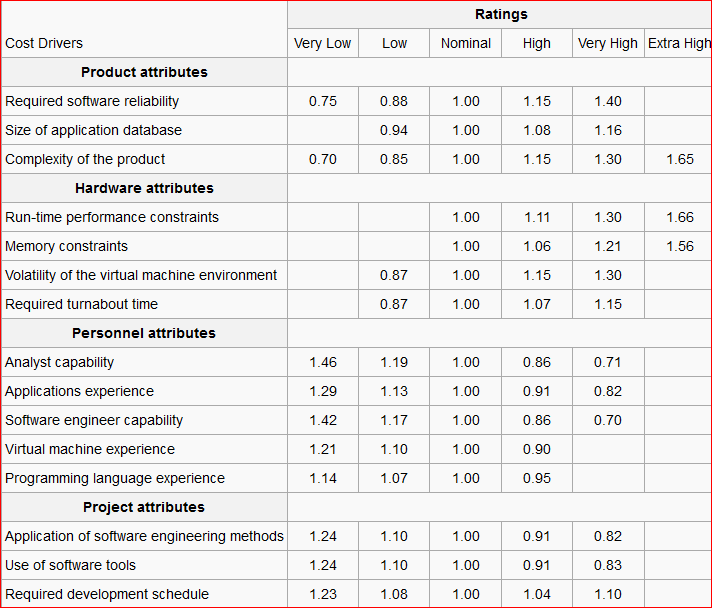
**Hardware Constraints:** This factor accounts for the restrictions or requirements imposed by the target hardware or platform. It is categorized into three levels: Low, Nominal, and High, depending on the hardware's impact on the project.

**Personnel Experience:** The model considers the experience level of the development team. It ranges from "Very Low" to "Very High" and reflects the team's collective expertise and familiarity with the project domain.

Intermediate COCOMO incorporates various multiplier values based on these factors to arrive at estimates for project effort, development time, and staffing requirements. The core formulas include power-law equations that capture the nonlinear relationships between these factors.

The final estimates are typically expressed in terms of:

* ***Effort (person-months):*** This represents the amount of work or labor required for project completion. It is a fundamental measure of the project's magnitude.
* ***Development Time (months):*** This indicates the duration needed to complete the project. It is often a critical parameter in project scheduling and planning.
* ***Number of Personnel:*** This reflects the team size required to complete the project within the estimated time frame. Staffing decisions play a crucial role in project management.

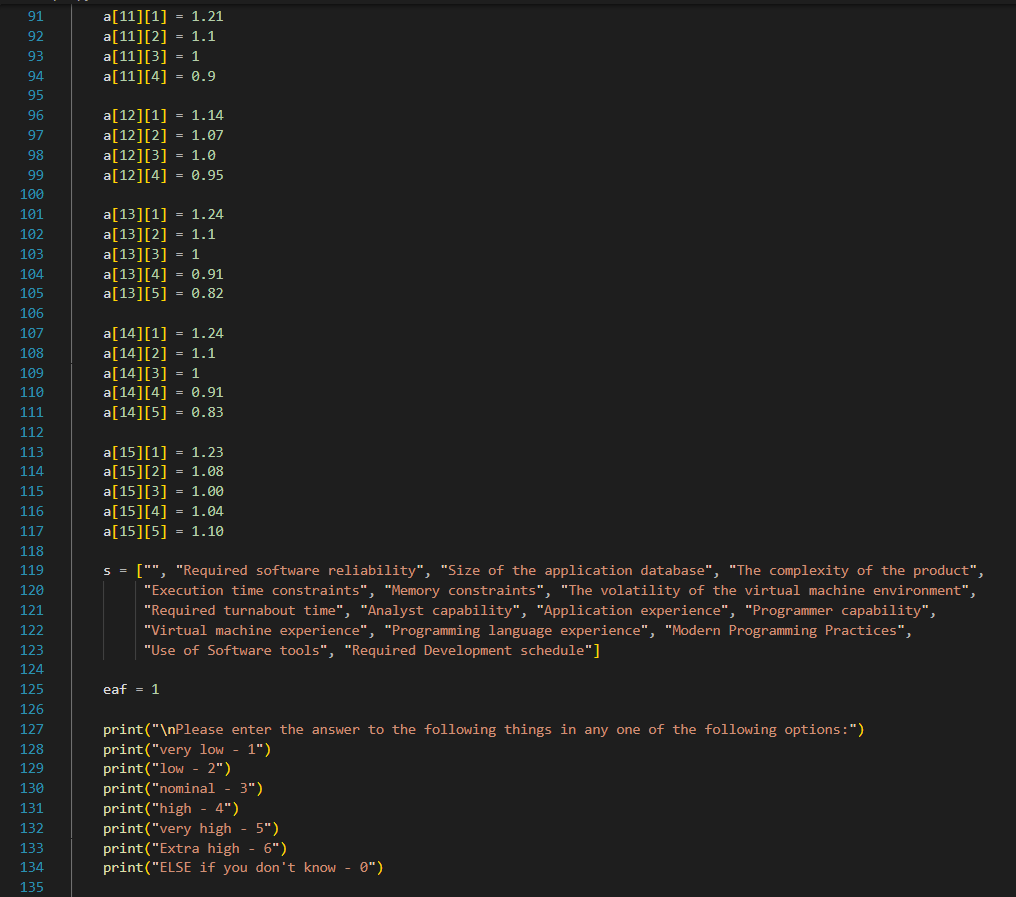


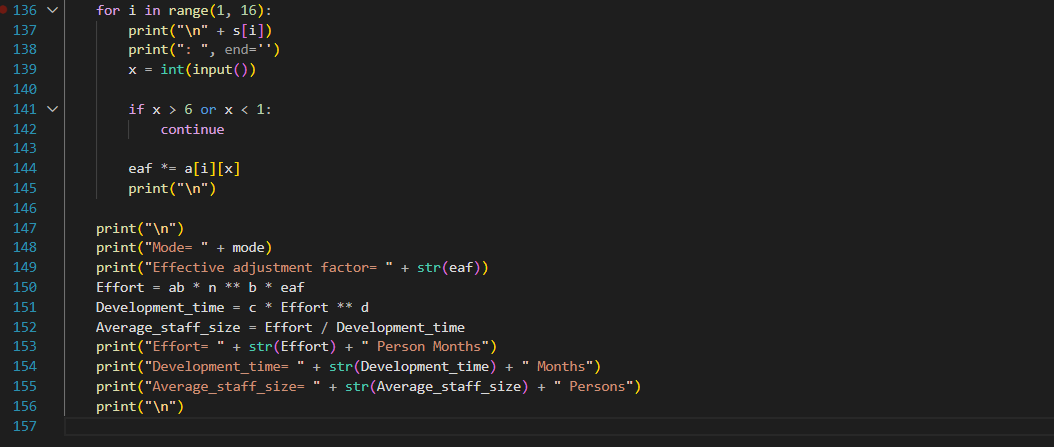
By using Intermediate COCOMO, software project managers and stakeholders can make more informed decisions about resource allocation, budgeting, and project scheduling. It helps in achieving a better understanding of the trade-offs between project size, complexity, team expertise, and available resources. Moreover, Intermediate COCOMO enhances the accuracy of software project cost estimation, reducing the risks associated with underestimation or overestimation of project parameters.

In summary, Intermediate COCOMO is a valuable tool in the software development industry, allowing organizations to create realistic and data-driven plans for successful project execution. It bridges the gap between the simplicity of the basic COCOMO model and the intricacies of real-world software development projects, making it an essential framework for project management and cost control.

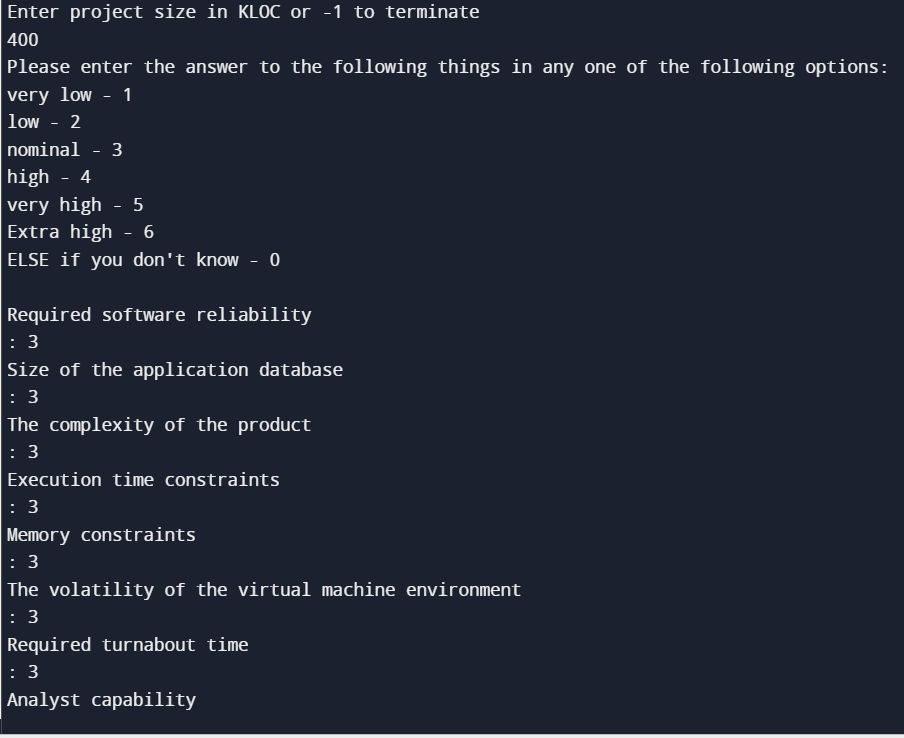
### CODE:

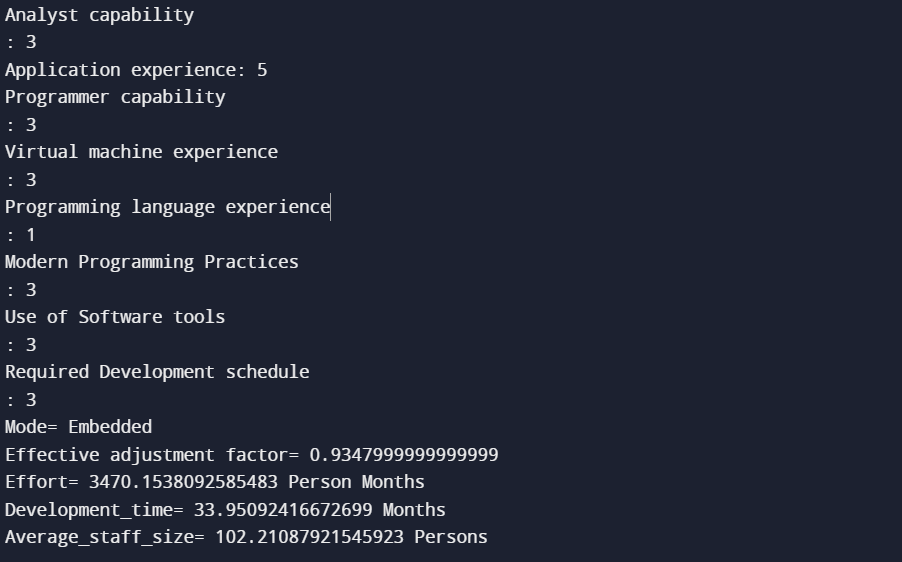
### 





### OUTPUT OF THE PROGRAM:





## LEARNINGS:

### By conducting this experiment, we have gained a practical understanding of how to implement intermediate COCOMO model in Python.

EXPERIMENT 5

**To write a program to implement Detailed COCOMO Model.**

## THEORY:

The Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver’s impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed COCOMO, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System design
3. Detailed design
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

**Algorithm:**

As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design document (DD%), code (C%) and integration (I%) to be modified are estimated.

Then, an adjustment factor, A, is calculated by means of the following equation:

A = 0.4 DD + 0.3 C + 0.3 I

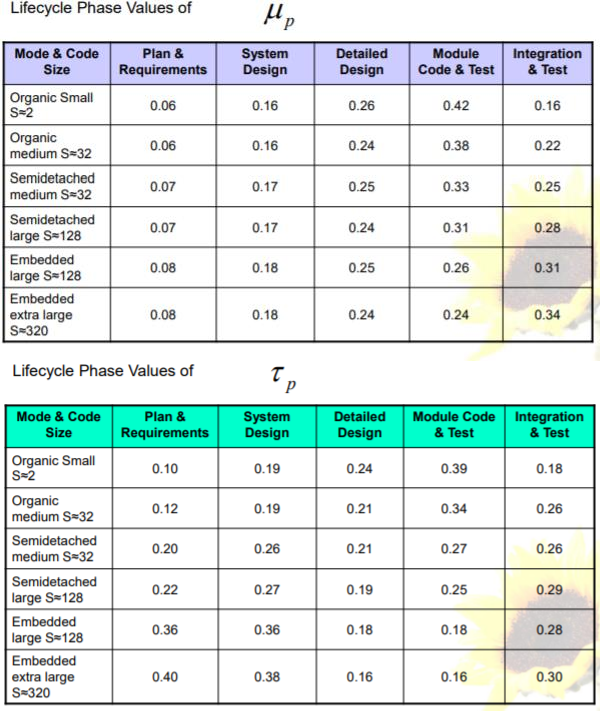
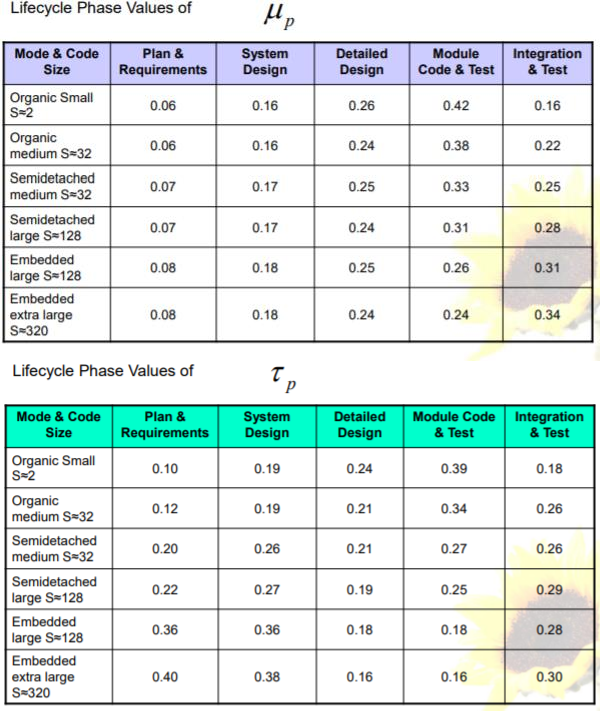
The size equivalent is obtained as :

S (equivalent) = (S x A) / 100

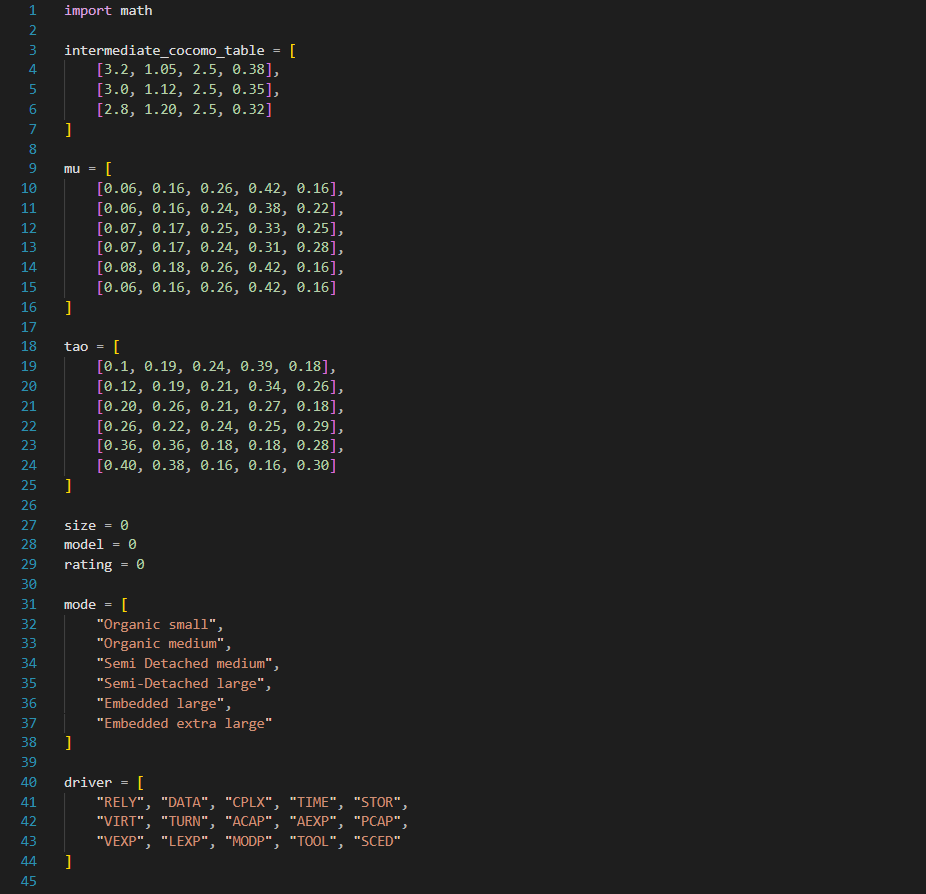
Ep = µpE

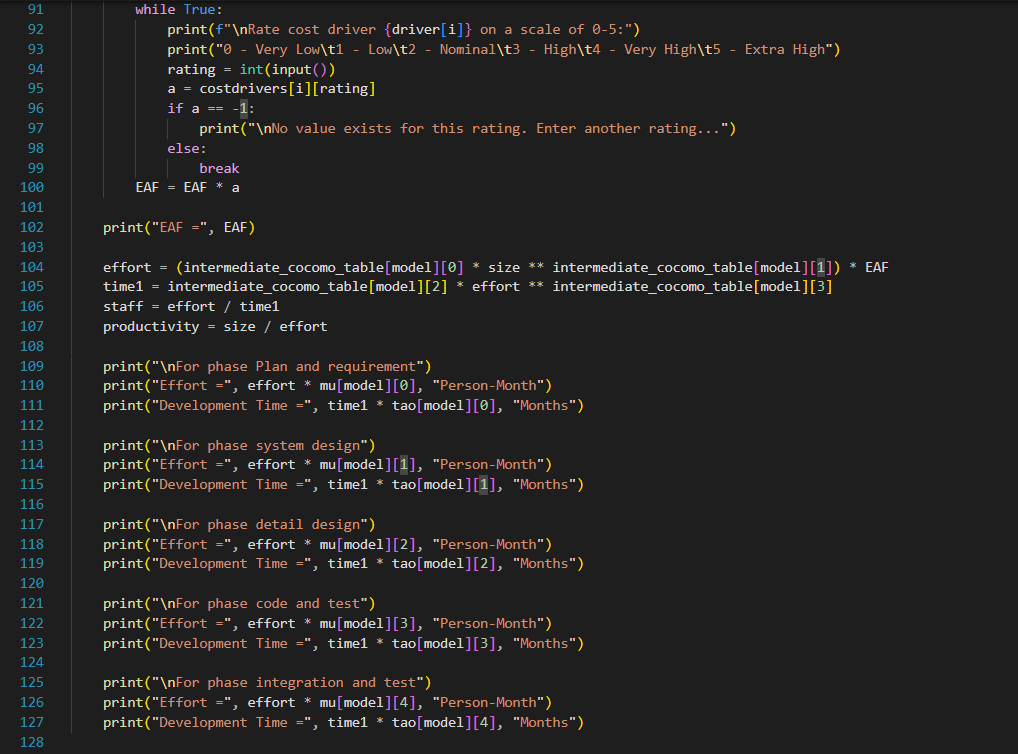
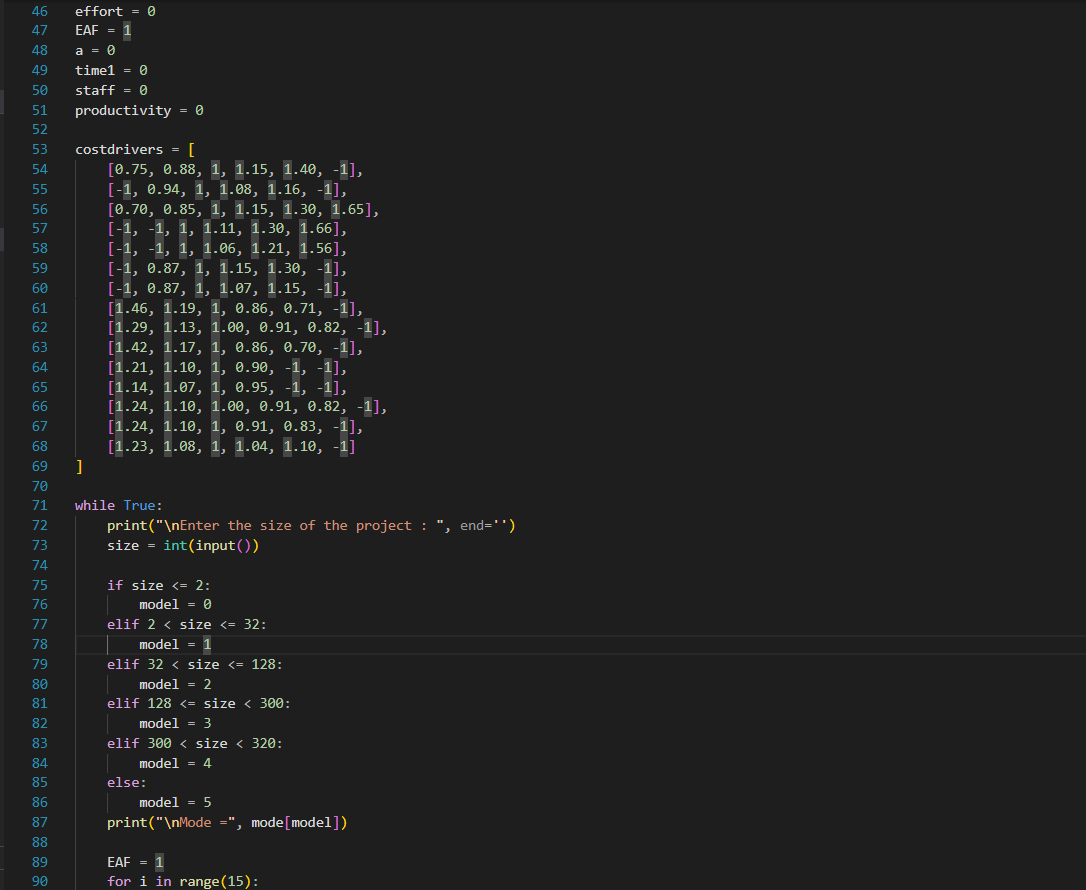
Dp = ͳpD

After calculating the total effort and duration, effort and duration for each phase is calculated using the two tables given below:

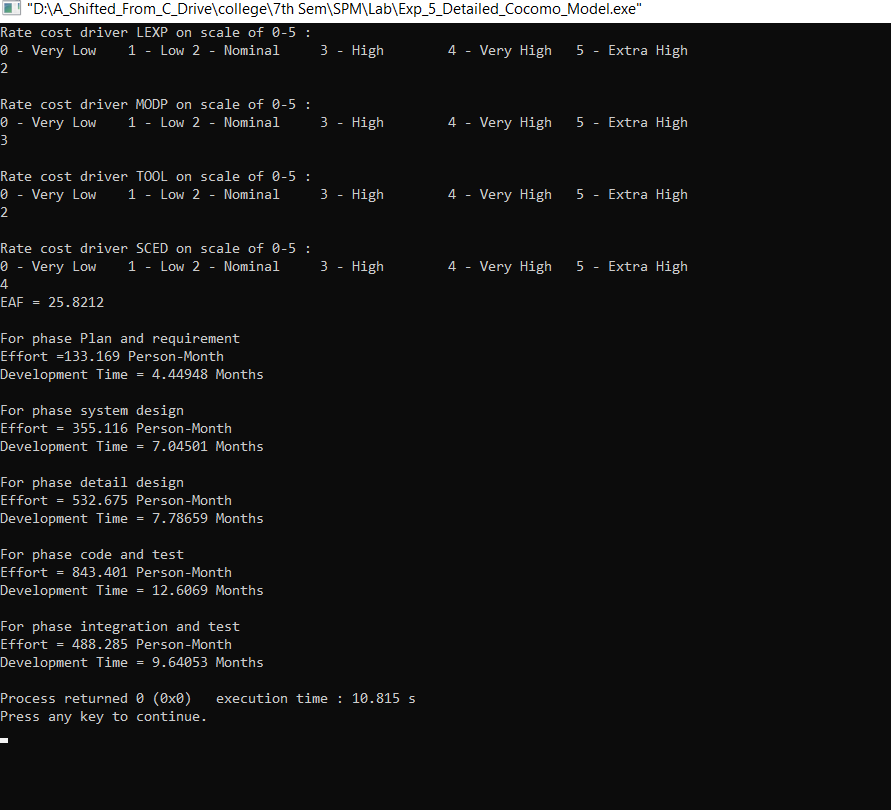
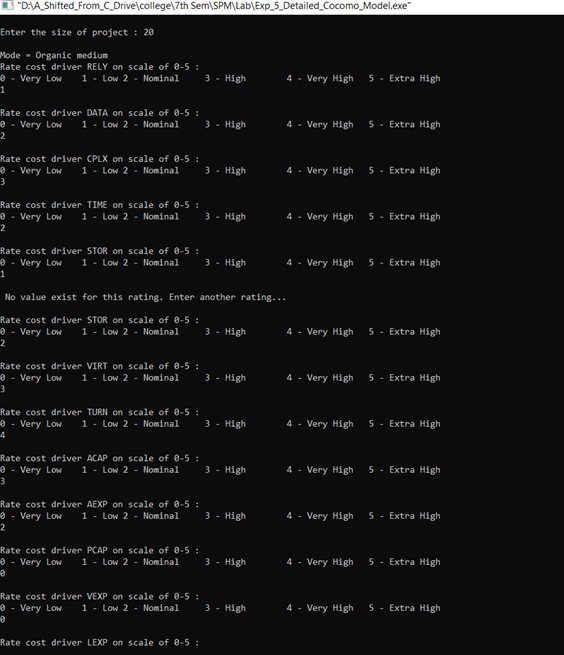
 

## CODE:





## OUTPUT:



## LEARNINGS:

### By conducting this experiment, we have gained a practical understanding of how to implement detailed COCOMO model in Python.

EXPERIMENT 6

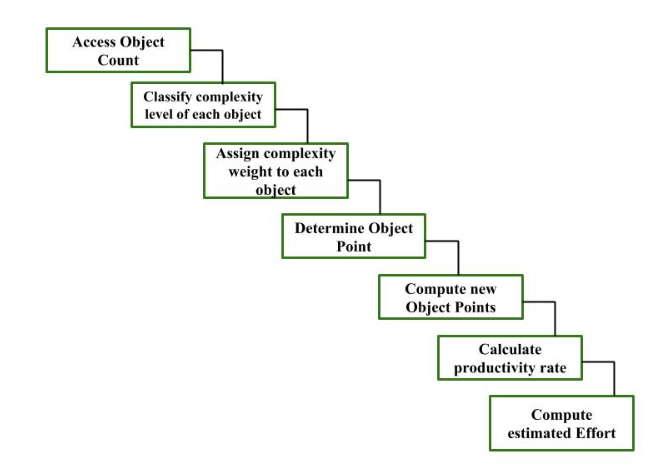
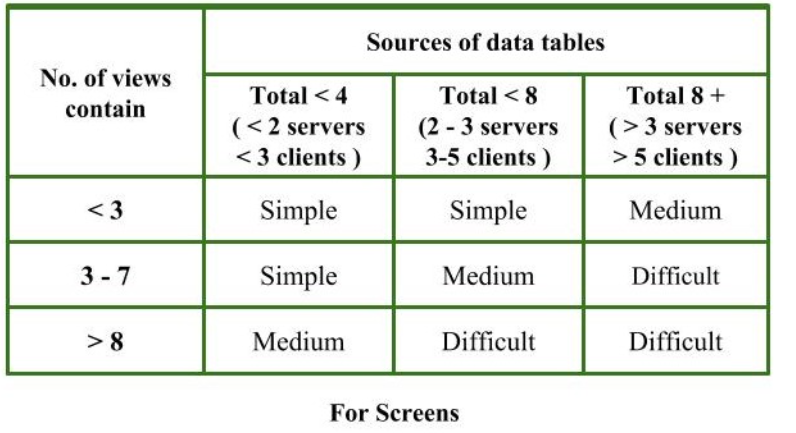
**Write a program to implement** **Application Composition Estimation Model for effort estimation.**

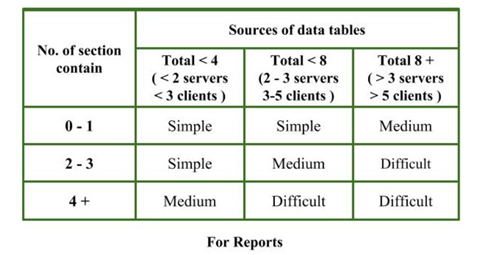
**THEORY:** In this model size is first estimated using *Object Points*. Object Points are easy to identify and count. Object Points defines screen, reports, third generation (3GL) modules as objects. Object Point estimation is a new size estimation technique but it is well suited in *Application Composition Sector.*

**ALGORITHM:**

**Step-1: Access Object counts**Estimate the number of screens, reports and 3GL components that will comprise this application.

**Step-2: Classify complexity levels of each object**We have to classify each object instance into simple, medium and difficult complexity level depending on values of its characteristics.Complexity levels are assigned according to the given table

****



**Step-3: Assign complexity weights to each object**The weights are used for three object types i.e., screens, reports and 3GL components.  
Complexity weight are assigned according to object’s complexity level using following table

**Step-4: Determine Object Points**Add all the weighted object instances to get one number and this is known as *object point count.*

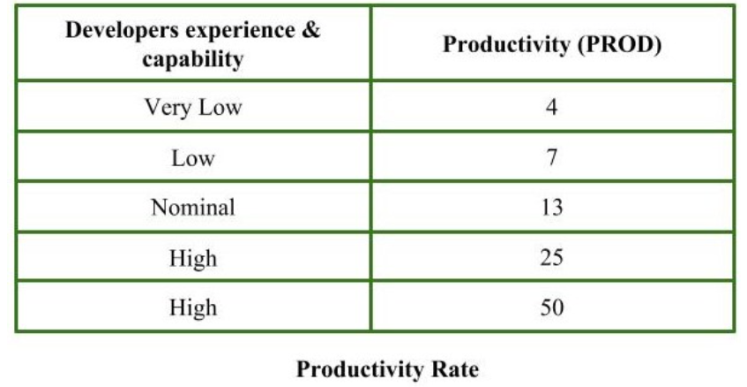
**Object Point**

= ∑ (number of object instances) \* (Complexity weight of each object instance)

**Step-5: Compute New Object Points (NOP)**We have to estimate the %reuse to be achieved in a project.  
Depending on % reuse

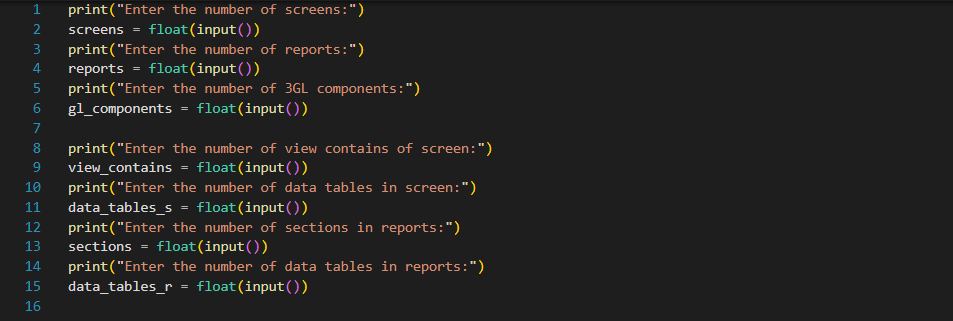
NOP = [(object points) \* (100 - % reuse)]/100

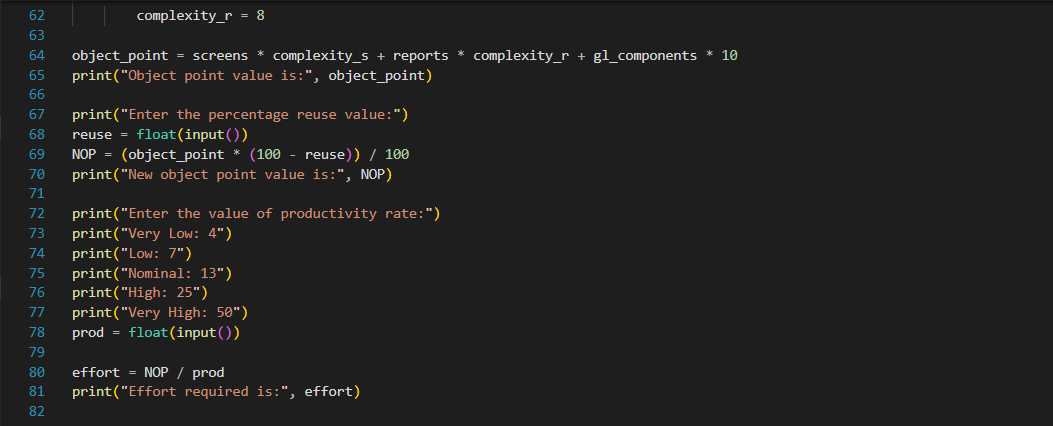
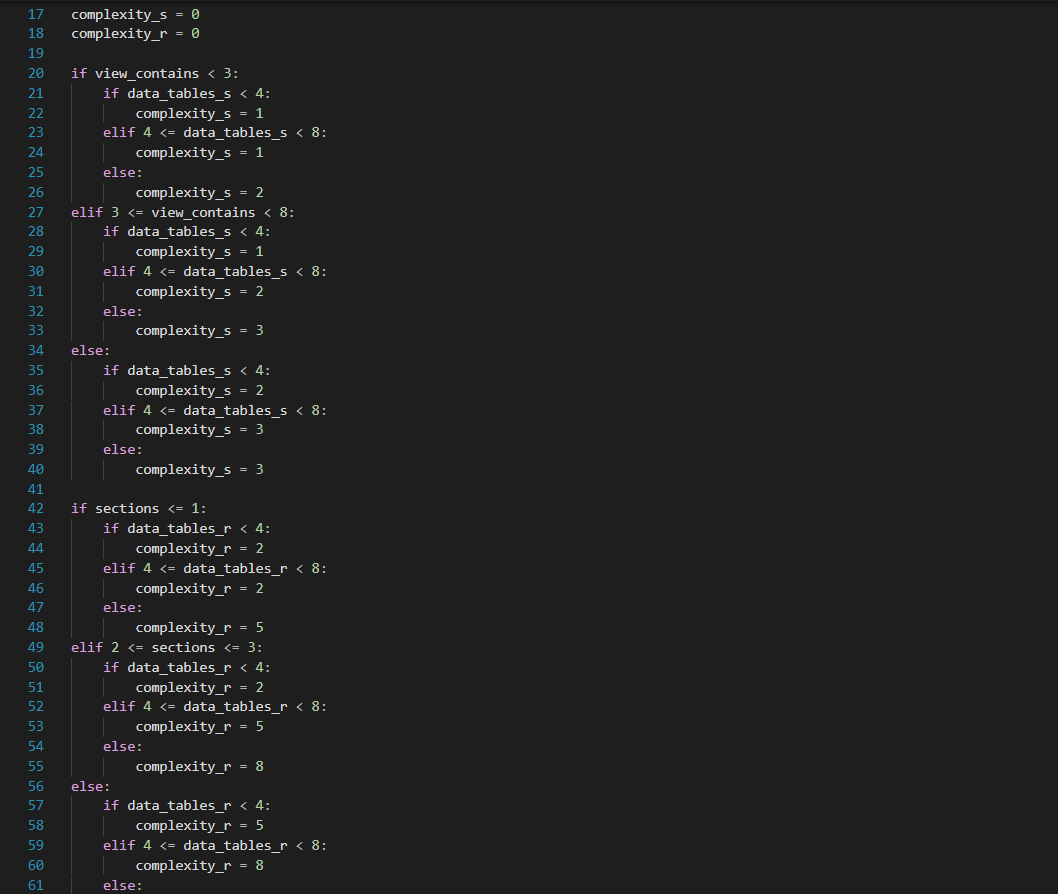
**Step-6: Calculate Productivity rate (PROD)***Productivity rate* is calculated on the basis of information given about developer’s experience and capability.

****

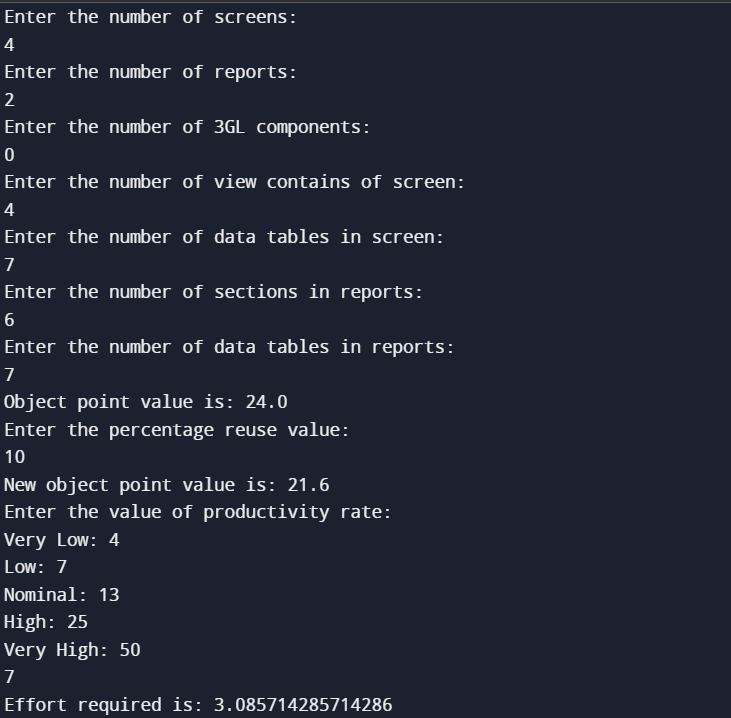
**Step-7: Compute the estimated Effort***Effort*to develop a project can be calculated as Effort = NOP/PROD

**CODE:**





**OUTPUT:**



**LEARNINGS:**

We have successfully learnt the way to implement Application Composition Estimation Model for effort estimation in python. The Application Composition Estimation Model gives much better approximate estimate of the project parameters as compared to Basic and Intermediate models

EXPERIMENT 7

**Write a program to implement Early Design Model and calculate the effort for the development.**

**THEORY:**

It is used at the Stage – II in COCOMO -II models and supports estimation in early design stage of project. Base equation used in COCOMO – II models is as follows –

PMnominal = A \* (size)B

where PMnominal = Effort for the project in person months

A = constant representing the nominal productivity where A = 2.5

B = Scale Factor

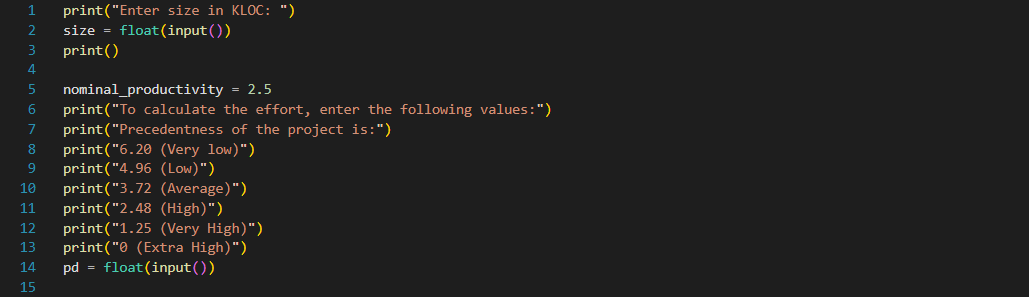
Size = size of the Software

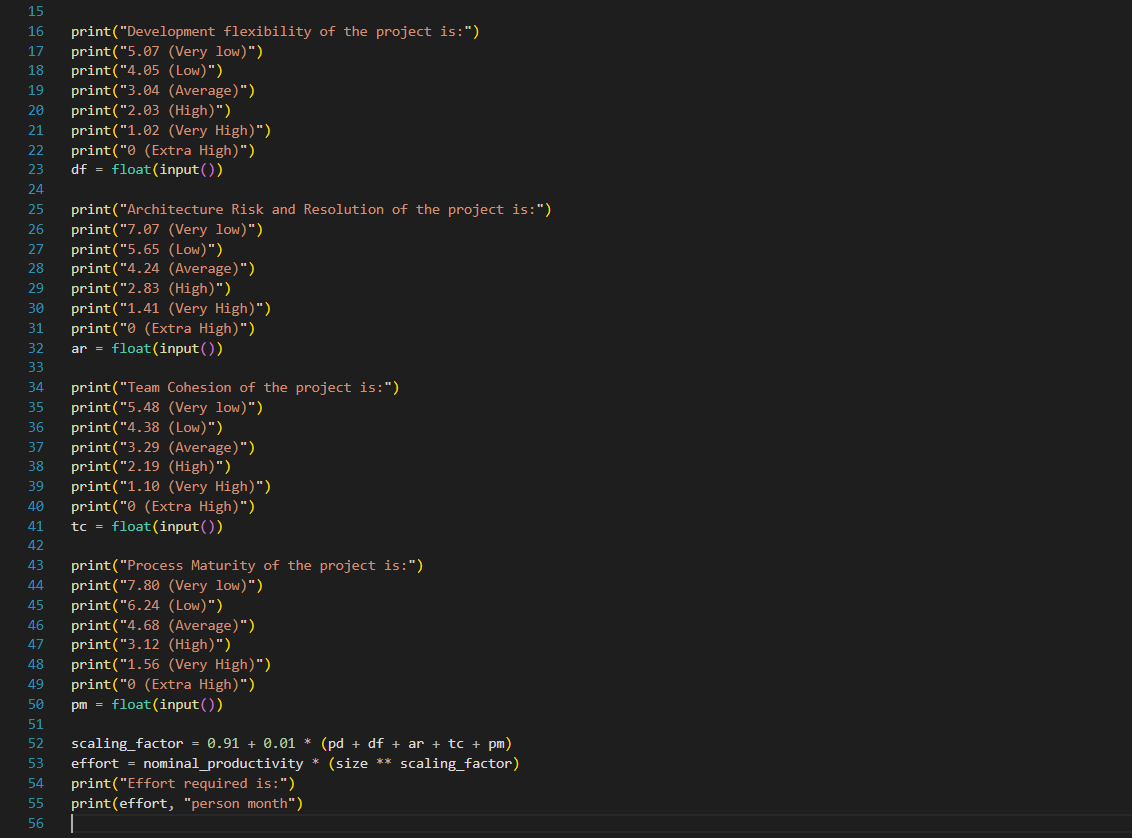
The early design model uses Unadjusted Function Points (UFP) as measure of size. This model is used at the early stages of software project when there is not enough information available about size of product which has t be developed, nature of target platform and nature of employees to be involved in development of projector detailed specifications of process to be used. This model can be used in Application Generator, System Integration, or Infrastructure Development Sector.

If B = 1.0, there is linear relationship between effort and size of product. If the value of B is not equal to 1, there will be non-linear relationship between size of product and effort. If B < 1.0, rate of increase of effort decreases as the size of product increases. If B > 1.0, rate of increase of effort increase as the size of product is increased.

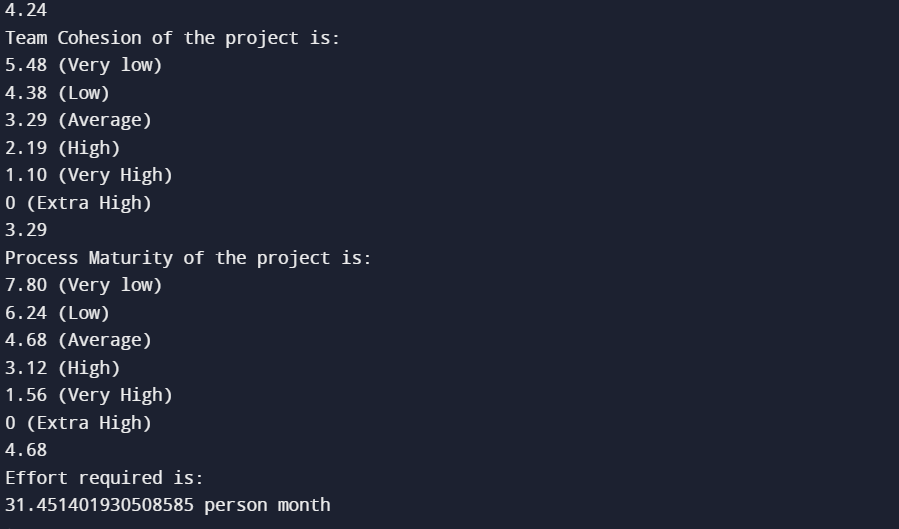
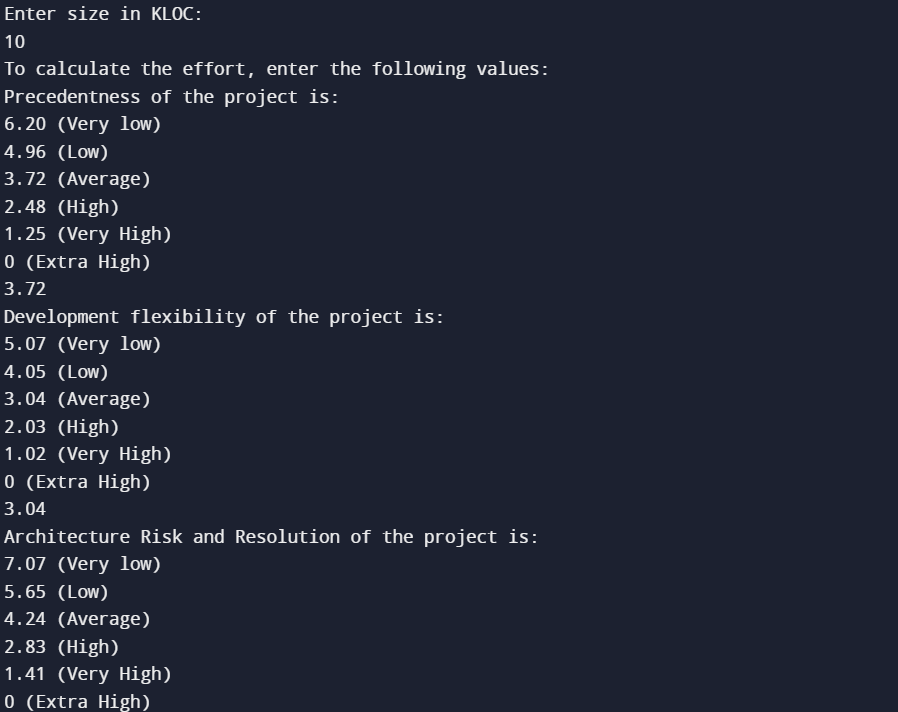
This is due to growth of interpersonal communications and overheads due to growth of large system integration. Application composition model assumes value of B equal to 1. But Early Design Model assumes value of B to be greater than 1.

**CODE:**

****



**OUTPUT :**



**LEARNINGS:**

We have successfully learnt the way to implement Early Design Model and calculate the effort for the development. in python.