Constructive Cost Model (COCOMO)

AIM: TO WRITE A PROGRAM IMPLEMENTING THE COCOMO MODEL IN C/C++.

Introduction

The Constructive Cost Model (COCOMO) is a procedural software cost estimation model developed by Barry W. Boehm. The model parameters are derived from fitting a regression formula using data from historical projects (61 projects for COCOMO 81 and 163 projects for COCOMO II).
COCOMO is used to estimate size, effort and duration based on the cost of the software.
Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of source lines of code (SLOC, KLOC).

COCOMO applies to three classes of software projects:

- Organic projects - "small" teams with "good" experience working with "less than rigid" requirements
- Semi-detached projects - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements
- Embedded projects - developed within a set of "tight" constraints. It is also combination of organic and semi-detached projects. (hardware, software, operational)

Software Project	Ab	Bb	Cb	Db
Organic	**2.4**	**1.05**	**2.5**	**0.38**
Semi-Detached	**3.0**	**1.12**	**2.5**	**0.35**
Embedded	**3.6**	**1.20**	**2.5**	**0.32**

Basic COMOMO Model (Table)
The basic equation of COCOMO Takes the form:

$$E = a_b KLOC^{b_b}$$
$$D = C_b E^{b_b}$$

where E is the effort applied in person-months, D is the development time in chronological months and KLOC is the estimated number of delivered lines of code for the project (express in thousands). The coefficients ab and cb and the exponents bb and db are given in Table 1.

Practical-7

21-Oct-2017

The basic model is extended to consider a set of "cost driver attributes" [BOE81] that can be grouped into four major categories:

1. Product attributes
    a. required software reliability
    b. size of application data base
    c. complexity of the product
2. Hardware attributes
    a. run-time performance constraints
    b. memory constraints
    c. volatility of the virtual machine environment
    d. required turnaround time
3. Personnel attributes
    a. analyst capability
    b. software engineer capability
    Applications experience
    d. virtual machine experience
    e. programming language experience
4. Project attributes
    a. use of software tools
    b. application of software engineering methods
    c. required development schedule

Code

```
#include <iostream>
#include <cmath>
float COCOMO[][4] = {
        2.4, 1.05, 2.5, 0.38,
        3.0, 1.12, 2.5, 0.35,
        3.6, 1.20, 2.5, 0.32
};
int main() {
    float loc;
    int type = 0;
    float E, D;
    std::cout << "Enter lines of code in 1000s: ";
    std::cin >> loc;
    if(loc > 50)
        type = 1;
    if(loc > 300)
        type = 2;

    E = COCOMO[type][0] * pow(loc, COCOMO[type][1]);
    D = COCOMO[type][2] * pow(E, COCOMO[type][3]);

    std::cout << "\nProject is ";
    switch(type) {
        case 0: {
            std::cout << "Organic";
            break;
        }
        case 1: {
            std::cout << "Semi detached";
            break;
        }
        case 2: {
            std::cout << "Embedded";
            break;
        }
    }
    std::cout << "\nE = " << E << "\nD = " << D << std::endl;

    return 0;
}
```

Output

```
Anurag@Jarvis MINGW64 /h/College stuff/College Stuff.Academic/College Stuff.Academic.Semesters/College.Stuff.A
cademic.Semesters.YEAR_3/SEM 5/SE/SE_LAB/Code (master)
$ g++ -o COCOMO cocomo.cpp

Anurag@Jarvis MINGW64 /h/College stuff/College Stuff.Academic/College Stuff.Academic.Semesters/College.Stuff.A
cademic.Semesters.YEAR_3/SEM 5/SE/SE_LAB/Code (master)
$ ./COCOMO.exe
Enter lines of code in 1000s: 1200

Project is Embedded
E = 17836.9
D = 57.3274
```

Learning

We implemented COCOMO model by taking the number of lines of code as an input and producing the metrics. COCOMO is well defined, and because it doesn't rely upon proprietary estimation algorithms, it offers these advantages to its users: estimates are more objective and repeatable than estimates made by methods relying on proprietary models and it can be calibrated to reflect your software development environment, and to produce more accurate estimates.