# SOFTWARE ENGINEERING

# (MC – 310)

# ASSIGNMENT – 4

**AIMAN SIDDIQUA** **2K18/MC/008**

---

**Ques. Explain all the levels of the COCOMO model. Some very large software projects involve writing millions of lines of code. Explain why the effort estimation models, such as COCOMO, might not work well when applied to very large systems. Assume that the size of an organic software product has been estimated to be 32,000 lines of code. Determine the effort required to develop the software product and the nominal development time.**

**Solution.**

**BASIC COCOMO MODEL**

The basic COCOMO model provides an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

$$\text{Effort} = a_1*(\text{KLOC})\ a_2\ \text{PM}$$
$$\text{Tdev} = b_1*(\text{efforts})b_2\ \text{Months}$$

where,

**KLOC:** Estimated size of the software product indicate in Kilo Lines of Code,

**$a_1$, $a_2$, $b_1$, $b_2$:** Constants for each group of software products,

**Tdev:** Estimated time to develop the software, expressed in months,

**Effort:** Total effort required to develop the software product, expressed in person months (PMs).

**INTERMEDIATE MODEL**

The basic COCOMO model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

**DETAILED COCOMO MODEL**

Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost drivers effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed COCOMO, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

In the case of large systems, where there will be millions of lines of codes, coding is only a small fraction of the total effort. LOC becomes a minor factor in project estimation. The estimation models may not work well when there is ambiguity in counting codes. Effort estimation depends on various factors such as requirements, plans, people, etc. So far on very large systems, the effort estimation models such as COCOMO might not work well.

Effort = x1 × (KLOC)^x2
KLOC = The estimated size of the software product expressed in Kilo lines of source code.
x1, x2, y1, y2 are constants for each category of Software products.
Effort = 2.4(KLOC)^1.05

Doing the substitution, we have:
KLOC = 32000/1000 = 32
Effort =2.4(32) ^1.05 = 91.331 = 91 pm (approx.)

## Ques. What is software quality? Discuss software quality attributes.

**Solution**. Software quality product is defined in terms of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc. for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Software quality attributes are:

- **Portability**: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

- **Usability**: A software product has better usability if various categories of users can easily invoke the functions of the product.

- **Reusability**: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

- **Correctness**: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

- **Maintainability**: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

### Ques. Explain why the intangibility of software systems poses special problems for software project management.

**Solution.** Software cannot be inspected like shipbuilding or a civil engineering project in which it is visible which parts of the structure is unfinished. Software is intangible. That means, it cannot be seen or touched.
Some of the important goals of software management are delivering the software in the proposed time, budget and to meet all the requirements of the customer. But the software being intangible, it is difficult for the software project managers to tell exactly what the status of the development is. They cannot see the progress by simply looking at the artifact that is being constructed. Rather they rely on others to produce evidence that they can use to review the progress of the work.

### Ques. Explain the SEI Capability Maturity Model (CMM).

**Solution.** The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.
The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.
CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center promoted by the U.S. Department of Defense (DOD).

The 5 levels of CMM are as follows:

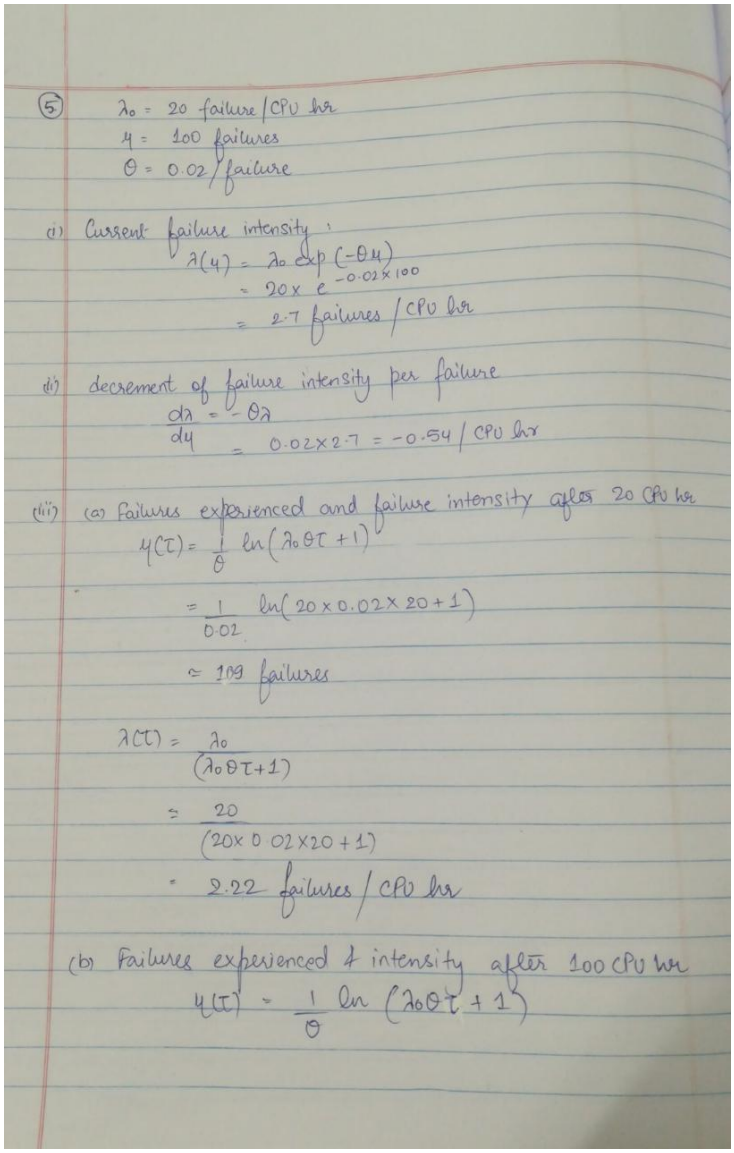
- **Initial** - Processes unpredictable, poorly controlled and reactive
- **Repeatable/Managed** - Processes characterized for projects and is often reactive
- **Defined** - Processes characterized for the organization and is proactive
- **Quantitatively Managed** - Processes measured and controlled.
- **Optimizing** - Focus on process improvement.

**Ques. Assume that the initial failure intensity is 20 failures/CPU hr. The failure intensity decay parameter is 0.02/failure. We have experienced 100 failures up to this time.**

i. **Determine the current failure intensity.**
ii. **Find the decrement of failure intensity per failure.**
iii. **Calculate the failures experienced and failure intensity after 20 and 100 CPU hrs. of execution.**
iv. **Compute additional failures and additional execution time required to reach the failure intensity objective of 2 failures/CPU hr.**

**Use Logarithmic Poisson execution time model of software reliability for the calculations.**

**Solution.**



⑤ $\lambda_0 = 20$ failure/CPU hr
$\mu = 100$ failures
$\theta = 0.02$/failure

(i) Current failure intensity:
$\lambda(\mu) = \lambda_0 \exp(-\theta\mu)$
$= 20 \times e^{-0.02 \times 100}$
$= 2.7$ failures/CPU hr

(ii) decrement of failure intensity per failure
$\frac{d\lambda}{d\mu} = -\theta\lambda$
$= 0.02 \times 2.7 = -0.54$/CPU hr

(iii) (a) Failures experienced and failure intensity after 20 CPU hr
$\mu(\tau) = \frac{1}{\theta} \ln(\lambda_0 \theta \tau + 1)$
$= \frac{1}{0.02} \ln(20 \times 0.02 \times 20 + 1)$
$\simeq 109$ failures

$\lambda(\tau) = \frac{\lambda_0}{(\lambda_0 \theta \tau + 1)}$
$= \frac{20}{(20 \times 0.02 \times 20 + 1)}$
$= 2.22$ failures/CPU hr

(b) Failures experienced & intensity after 100 CPU hr
$\mu(\tau) = \frac{1}{\theta} \ln(\lambda_0 \theta \tau + 1)$

$$= \frac{1}{0.02} \ln(20 \times 0.02 \times 100 + 1)$$

$$= 186 \text{ failures}$$

$$\lambda(\tau) = \frac{\lambda_0}{(\lambda_0 \tau \theta + 1)}$$

$$= \frac{20}{(20 \times 0.02 \times 100 + 1)}$$

$$= 0.4878 \text{ failure / CPU hr}$$

(iv) Additional failures ($\Delta\mu$) required to reach failure intensity objective of 2 failure/CPU hr.

$$\Delta\mu = \frac{1}{\theta} \ln\left(\frac{\lambda_P}{\lambda_F}\right) = \frac{1}{0.02} \ln\left(\frac{27}{2}\right)$$

$$= 15 \text{ failures.}$$

$$\Delta\tau = \frac{1}{\theta}\left[\frac{1}{\lambda_f} - \frac{1}{\lambda_P}\right] = \frac{1}{0.02}\left[\frac{1}{2} - \frac{1}{27}\right]$$

$$= 6.5 \text{ CPU hr.}$$