

SOFTWARE MAINTENANCE AND METRICS

Software Maintenance

- Terminology

- u **Software Maintenance**

- consists of the activities required to keep a software system operational and responsive after it is accepted and placed into production.

- u **Software Maintainer**

- person whose mission is to support existing software systems

CAUSES FOR MAINTENANCE

❑ **Market Conditions**

- Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.

❑ **Client Requirements**

- Over the time, customer may ask for new features or functions in the software.

❑ **Host Modifications**

- If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.

❑ **Organization Changes**

- If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Maintenance Categories

- u Corrective maintenance
- u Adaptive maintenance
- u Perfective maintenance
- u Preventive maintenance

CORRECTIVE MAINTENANCE

- u Deals with the repairing faults or defects being faced regular system functions.
- u A defect can result due to errors in software design, logic and coding.
- u Design errors occur when changes made to the software are incorrect, incomplete, wrongly communicated, or the change request is misunderstood.
- u Logical errors result from invalid tests, incorrect implementation of design specifications, faulty logic flow, or incomplete test of data.

- u All these errors, referred to as residual errors, prevent the software from conforming to its agreed specifications.
- u The need for corrective maintenance is usually initiated by bug reports drawn by the users.
- u The approach in corrective maintenance is to locate the original specifications in order to determine what the system was originally designed to do.
- u In the event of a system failure due to an error, actions are taken to restore the operation of the software system.
- u Corrective maintenance accounts for 20% of all the maintenance activities.

PERFECTIVE MAINTENANCE

- u Deals with implementing new or changed user requirements.
- u Involves making functional enhancements to the system in addition to the activities to increase the system's performance even when the changes have not been suggested by faults.
- u Includes all efforts to improve the quality of the software.
- u Includes restructuring code, creating and updating documentation, improving reliability or efficiency.
- u Perfective maintenance accounts for 50%, that is, the largest of all the maintenance activities.

ADAPTIVE MAINTENANCE

- u Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system.
- u Adaptive maintenance consists of adapting software to changes in the environment such as the hardware or the operating system.
- u The term environment in this context refers to the conditions and the influences which act (from outside) on the system i.e. business rules, work patterns, and government policies etc.
- u Adaptive maintenance accounts for 25% of all the maintenance activities.

PREVENTIVE MAINTENANCE

- u Preventive maintenance involves performing activities to prevent the occurrence of errors.
- u It tends to reduce the software complexity thereby improving program understandability and increasing software maintainability.
- u It comprises documentation updating, code optimization, and code restructuring.
- u Documentation updating involves modifying the documents affected by the changes in order to correspond to the present state of the system.
- u Code optimization involves modifying the programs for faster execution or efficient use of storage space.
- u Preventive maintenance is limited to the maintenance organization only and no external requests are acquired for this type of maintenance.
- u Preventive maintenance accounts for only 5% of all the maintenance activities.



Software metrics

- A software metric is a measure of software characteristics which are quantifiable or countable.
- Software metrics are important for many reasons, including:
 1. Measuring software performance,
 2. Planning work items,
 3. Measuring productivity, and many other uses.




Function-Oriented Metrics

- It use a measure of functionality delivered by the application as a normalization value.
- Since 'functionality' cannot be *measured directly*, it must be derived *indirectly using other direct measures*
- Function Point (FP) is widely used as function oriented metrics.
- FP is based on characteristic of *Software information domain*.
- FP is programming language independent.
- Function points are computed by completing the table shown in following Figure.
- Five **information domain characteristics** are determined and counts are provided in the appropriate table location.

FP- Five information domain characteristics

Measurement parameter	Weighting factor					
	Simple		Average		Complex	
Number of user inputs	3 x _		4 x _		6 x _	
Number of user outputs	4 x _		5 x _		7 x _	
Number of user inquiries	3 x _		4 x _		6 x _	
Number of files	7 x _		10 x _		15 x _	
Number of external interfaces	5 x _		7 x _		10 x _	
	Simple Total		Average Total		Comple x total	
Count Total						

- 
- **Number of user inputs** - Each user input that provides distinct data to the software is counted
 - **Number of user outputs** - Each user output that provides information to the user is counted. Output refers to reports, screens, error messages, etc
 - **Number of user inquiries** - An inquiry is defined as an *on-line input* that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted. (i.e. search index, google search)
 - **Number of files** - Each logical master file (i.e. large database or separate file) is counted.
 - **Number of external interfaces** - All machine readable interfaces (e.g., data files on storage media) that are used to *transmit information to another system* are counted.

- To compute function points (FP), the following relationship is used:

$$\underline{FP = count\ total\ [0.65 + 0.01 \sum(Fi)]}$$

The Fi ($i = 1$ to 14) are "complexity adjustment values".

- Each of these values measure on scale based ranges from 0 (not important or applicable) to 5 (absolutely essential)
- Once function points have been calculated, they are used in a manner analogous to LOC as a way to normalize measures for software productivity, quality, and other attributes:
 - Errors per FP.
 - Defects per FP.
 - \$ per FP.
 - Pages of documentation per FP.
 - FP per person-month.



Questions for complexity adjustment values

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?


Constructive Cost Model (COCOMO)


- 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 often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.





Constructive Cost Model (COCOMO)


- It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make 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. It is measured in the units of time such as weeks, months.

- 
- It can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.
 - **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.

- 
- **Semi-detached** – A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, 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.

- 
- **Embedded** – A software project with 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.

- 
- It start with an initial estimate determined by using the static single-variable model equations, which depend on size, and then adjusting the estimates based on other variables.
 - This approach implies that size is the primary factor for cost; other factors have a lesser effect.
 - The basic steps in this model are:


- 
- 1. Obtain an initial estimate of the development effort from the estimate of thousands of delivered lines of source code (KLOC).
 - 2. Determine a set of 15 multiplying factors from different attributes of the project.
 - 3. Adjust the effort estimate by multiplying the initial estimate with all the multiplying factors.


- The initial estimate (also called *nominal estimate*) is determined by an equation of the using KLOC as the measure of size.
- To determine the initial effort *E_i in person-months* the equation used is of the type:

$$E_i = a * (KLOC)^b.$$


- The constants a and b for different systems are:

System	a	b
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

- 
- There are 15 different attributes, called *cost driver attributes*, that determine the multiplying factors.
 - These factors depend on product, computer, personnel, and technology attributes (called *project attributes*).

- 
- Each cost driver has a rating scale, and for each rating, a multiplying factor is provided.
 - The rating scale is very low, low, nominal, high, and very high (and in some cases extra high).
 - The attributes and their multiplying factors for different ratings are shown in following Table:

Cost Drivers	<i>Rating</i>				
	Very Low	Low	Nom- inal	High	Very High
Product Attributes					
RELY, required reliability	.75	.88	1.00	1.15	1.40
DATA, database size		.94	1.00	1.08	1.16
CPLX, product complexity	.70	.85	1.00	1.15	1.30
Computer Attributes					
TIME, execution time constraint			1.00	1.11	1.30
STOR, main storage constraint			1.00	1.06	1.21
VITR, virtual machine volatility		.87	1.00	1.15	1.30
TURN, computer turnaround time		.87	1.00	1.07	1.15
Personnel Attributes					
ACAP, analyst capability	1.46	1.19	1.00	.86	.71
AEXP, application exp.	1.29	1.13	1.00	.91	.82
PCAP, programmer capability	1.42	1.17	1.00	.86	.70
VEXP, virtual machine exp.	1.21	1.10	1.00	.90	
LEXP, prog. language exp.	1.14	1.07	1.00	.95	
Project Attributes					
MODP, modern prog. practices	1.24	1.10	1.00	.91	.82
TOOL, use of SW tools	1.24	1.10	1.00	.91	.83
SCHED, development schedule	1.23	1.08	1.00	1.04	1.10

- 
- The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF).
 - The final effort estimate, E , is obtained by multiplying the initial estimate by the EAF. That *is*,

$$E = EAF * Ei..$$

By this method, the overall cost of the project can be estimated.