What is software?

Ans.

Software is a program when it is executed the associated functionality must be satisfied.

Slw is a data structure used to manipulate the information.

Slw is a document used to maintain the operational guidelines of the program.

**: Slw is a combination of:

- code
- analysis, design
- testing
- guidelines to naive users.

Slw is a logical entity rather than a physical entity.

Characteristics of the Slw

1. It is developed or engineered but not manufactured.
2. Co/p is a logical entity.
3. It contains analysis, design, coding, testing, etc.
4. Co/p of manufacturing is a physical entity.
5. It contains analysis, design, manufacturing process.
6. Slw is a logical component & hlw is a physical component.

After implementing the design, if we get physical component as o/p, then that process is called manufacturing process, & if after implementation of design, the o/p we get is a logical component, then process is called development.
(ii) S/W doesn’t wear out, but it deteriorate.

When S/W undergoes deterioration, we have to re-engineer the product.

Note: Ideal curve: When customer requirements are static. Real curve: When customer requirements are dynamic.

H/W undergoes wear out, which means, at early stage of development the uncovered errors cause the failure rate of product high. This state is called burning state. After maintenance of product, the defects are removed from the product, so failure...
Rate decreases. Then, the product will be working in the specified levels. This phase is useful life phase. Over a period of time, due to environmental changes (dust, temp., etc.) the product failure rate increases. This phase is wear-out phase. When product undergoes wear-out, replace the component with new component.

Deterioration :-
• Slowly undergoes deterioration, which means user requirements are not static. Due to the new requirement changes, the product failure rate increases after.
• After maintenance, the defects will be minimized but due to the frequent changes, the product maintenance cost becomes twice or thrice of the development cost. This condition is called as deterioration.
• When slow deterioration, then re-engineer the slw.

(iii) Industry moving towards component-based development, but still slw is a custom-built:
• Component is an error-free code or reusable code or fully tested code.
• In the physical component development, the components are directly used in the process without changes (diff. IC’s)
• But in the slw development, the components are modified acc. to the problem objective.
A Software components are divided into 4 types:

- Off-the-shelf
- Fully experienced
- Partially experienced
- New components

- **Off-the-shelf**: components are extracted from the 3rd party libraries & used directly in the current project.

- Fully or Partially experienced components are extracted from the current organisation library & used in the project.

- The new components functionality are developed from the base line.

Software Applications:

- There are different application domains that are using the S/W to satisfy the objective within the given time period.

G) System Software:

- It is a program used to manage the other program e.g. compiler, OS, device drivers.

G) Real-time S/W:

- This S/W satisfies the high-level objectives of the real-world entities, e.g. weather forecasting, aircraft simulation. (At there is a latency in user action & S/W reaction, then it may cause serious damages)
(iii) Business S/w:-
It is used to generate the reports. Various programming languages are used in this domain, e.g. COBOL, OLAP Tools.

(iv) Engg 4 scientific S/w:-
It is used to perform the high level computation, e.g. CAD/CAM, MATLAB, NS2.

(v) Embedded S/w:-
This s/w is stored in the permanent memory of the devices (ROM).
When the device is in the operational state, then the s/w is executed, so the associated functionality must be satisfied.

(vi) Web S/w:-
By using the web s/w, we can develop user interfaces, e.g. web services, HTML, etc.

(vii) Artificial Intelligence:-
In this domain, symbolic computations are performed rather than numeric computation, e.g. knowledge representation, image processing, speech recognition, machine learning.
To perform the symbolic computation, functional programming languages are used, e.g. LISP (LISP Processing), Prolog (Logic Programming), Haskel, etc.

* To satisfy various high-level objectives, there is a need to develop efficient s/w. S/w engg. practices were required to develop the quality s/w.
S/w Engg -
the application of systematic, disciplined & quantifiable approach to develop, operate & maintain the s/w.

1. Layered Approach
2. Generic Approach

Layered Approach

![Diagram showing layers of development](#)

Layers to develop Quality S/w.

(i) The quality layer is a bedrock of the s/w. Quality is defined as confirmation to explicitly stated requirements. Quality is an indirect measurement, so it depends on defects.

\[
\text{Quality} \approx \frac{1}{\text{defect}}
\]

The uncovered errors becomes the defect during the operational state of the s/w.

(ii) Process layer focuses on key process areas i.e. the product, project, people, process to trace the cost, effort & schedule. (There is no measurement related to people.)
(iii) Method layers focus on the framework & process model to develop the quality SW.

(iv) Tools layers focus on the automated & semi-automated tools present in the 4th generation techniques (focus on 4th generation language.)

**Generic Approach:**
Generic approach comprises of 3 phases of operation:

(i) **Definition → focus on “what”**: [e.g., what behaviour is expected from system, analysis]

(ii) **Development → focus on “how”**: [how to convert “what” info. gathering (4t hup & 4thp) of the system, designing, coding & testing.]

(iii) **Support → focus on “why”**: [for maintenance purposes (in operational mode), maintenance is corrective maintenance]

**Definition:**
This stage focuses on “what”, i.e., “what” info. is processed, “what” functionality or performance is designed & “what” behaviour is expected. “What” constraints are imposed & “what” validation criteria is used.

In this stage, requirements analysis & req. gathering activities are performed.

**Validation:** “Did we build the right system?”
The validation process is injected into the early stage of the SW development to minimize the maintenance effort & cost.

**Verification:** “Did we build the system rightfully”
Verification activity is performed in the testing
stage of the slwr development life cycle.

Development:
This stage focuses on "design," i.e., how to create the data structure, how to define the interfaces, how to translate the procedural description into machine-readable format, and how to define and implement the test cases.

In this stage, design, coding, and testing activities are performed.

Support:
After deploying product at the customer place, support stage is required to maintain the product from various side effects. These are 4 types of support present:-

(i) Correction: In this support, the uncovered errors are corrected, so corrective maintenance is required to cover the uncovered errors.

(ii) Adaption: This support requires adaptive maintenance to maintain the product due to the platform changes, e.g., due to CPU/ Memory/OS Upgradation.

(iii) Enhancement: This support is required when the functional requirement changes. Perfective maintenance is required to maintain the slwr from the functionality changes.

(iv) Preventance: This support uses preventive maintenance to maintain the slwr from frequent changes.
The maintenance effort is distributed as follows:

![Diagram showing distribution of maintenance effort]

Software Process

Process is a framework, there are different types of the framework (process models) are used to develop the quality S/W.

Software engineers institution define one assessment model to assess the S/W organisation based on their level of process development. The assessment model is named as CMM (Capability Maturity Model).

This model consists of 5 levels of assessment, i.e.:

(i) Level-0: Initial. (No standard model to develop the S/W)
(ii) Level-1: Repeatable (Management activities are defined)
(iii) Level-2: Defined
(iv) Level-3: Managed
(v) Level-4: Optimised

Initial:
In this level, adhoc procedures are used to develop the S/W; the process becomes chaotic. So success/failure depends on the individual
Repeatability:
In this level, management activities are defined based on previous project knowledge to trace the cost, schedule & effort of the project, no guarantee to deliver the product on time & in-budget.

Defined:
In this level, management & engg. activities are defined but not properly documented, no soln. for the anticipated situation.

Managed:
In this level, engg. & management activities are qualitatively documented & controlled. Due to lack of continuous improvement, min. quality will be assured.

Optimized:
Gain more profit with min. effort is the objective of the optimized level. In this level, innovative tools & techniques are used & continuously taking the qualitative feedback from the customers.

Process Models:
The classification of the process models are shown below:
Conventional Model:

By using the conventional process model, we can develop the complete operational product.

(i) Waterfall Model:

It contains the following activities:

1. **Analysis** → SRS → Design → Code → Testing → Support
2. System Engineering

System Engineering:

(i) Software is a subset of the system, so to develop the software, there is a need of analysing the existing system to finalise the goals of the product.

**Analysis**: In this stage, based on the goals of the product, different requirements are finalised. All of the right requirements are placed in the SRS document. SRS document consist goal of the software, functional requirement, non-functional requirements.
Design: In this stage, 4 activities are performed:

(i) Creating the data structure.
(ii) Creating the architecture.
(iii) Defining the interfaces.
(iv) Defining the component description.

After converting SRS document into the procedural description, the design document is prepared.

Code: In this stage, the procedural description is converted into machine-readable format by taking the appropriate language from 4GL.

Test: In this stage, different test cases are defined and implemented to cover the structural and functional aspects.

After deploying the product at the customer domain, there is a need to maintain the product from side effects.

Note: Waterfall model is suitable when the customer requirements are clear initially and customers can wait because in this model, the final product is available after a long schedule. It is not suitable to develop complex systems because it is very difficult to determine all requirements at a time.
Prototype Model

When the customer requirements are not clear, then prototype model is used to finalize the requirement. The framework of prototype model is to listen to the customer, quick design & testing mockup & customer evaluation mockup.

In this model, customer communication is allowed. Based on the customer requirements, the developer prepares the SRS document. This SRS is not final. Based on the SRS, the developer develops the prototype product. After demonstrating the prototype product, the developer accepts the feedback from the customer. When feedback consists of new requirements, the SRS document is modified, otherwise the corresponding SRS is finalized.

After finalizing the SRS document, if we still use the same framework to develop the final product, it is open-end/evolutionary prototype. After finalizing SRS, if we use different process model to develop the product is called 'throw away prototype model.'
RAD

(i) RAD model is used when the customers require more clarity & the project schedule is very short.

(ii) To develop the product in short schedule, there is a need of using the modularity concept. Before taking the modularity, the development team is divided into subteams that means efficient developers are put in the org.

(iii) Modularity means divide the project into smaller modules & develop them simultaneously.

(iv) Two factors
   - Cohesion
   - Coupling

   Cohesion is a major measure of to what extent the module is independent of other modules.
   - Coupling is a measure of what extent the module is depend of the other module.

The effective modulating maintaining the high cohesion & low coupling.

- With respect to Modularity concept, all the modules are developed simultaneously. After development, there is a need of
Integration to develop the final system.
The framework of RAD model is:

# Team #1  Team #2  Team #2

- Business Modelling
- Data Modelling
- Process Modelling
- Application Generation
- Test & Turnover

also known as component-based classic lifecycle.

- In Business modelling, the goals of the product are finalized.
- In the data modelling, diff. data objects are required to satisfy the goal of the product.
- In the process model, the info. flow is created by adding/deleting/modifying the data objects.
- In the application generation, the procedural description of problem statement (algo) is converted into machine language.
- In Test & Turnover stage, various test cases are implemented to cover the diff. env. & sys.

Note: The objective of the testing is to prove the system up to what extent the system.
At the end of test & turnover, we have an error-free module (components) to integrate all the components to develop the final product.
Note: RAD model maintaining less effort during testing phase, whereas in瀑布fall model, testing phase require more effort.
This model uses the classic life cycle framework to develop the product version by version (or step by step) to handle risk.

In the evolutionary process model, the final operational product will be available after some iterations. There are different frameworks employed under this category.
**WIN-WIN Spiral Model**

- Release by release customer satisfaction increases.

(i) In this model, customer involvement takes place, i.e., continuously taking the qualitative feedback from the customer.

(ii) The entry point in this model to develop the product is customer communication.

(iii) Based on the customer, e.g., SRS document is finalised.

(iv) During the planning stage, different parameters of the S/W undergo estimation, e.g., size, effort, schedule, cost, etc.

(v) In this model, risk impact is calculated during the risk analysis stage.

(vi) After taking the Risk impact, it is added to the project budget to develop the S/W on-time & in-budget.

(vii) In the engineering stage, the design practices are defined (Data Structure, Architecture, etc.)
(viii) After converting SRS into procedural description (Algol) by using diff tools & techniques, they develop the code.

(ix) After implementing different test cases, the product will be released at customer's site.

(x) After releasing the product, continuous feedback is accepted from the customer to improve the SW functionality & reliability.

(xi) In this model, customer satisfaction & developer satisfaction levels are high. So, it is also called as WIN-WIN Spiral model.

Component Based Development Model:

- In this model, framework, reusable components are extracted from the library & directly used in the project development. The framework of the component-based development is

![Diagram of Component Based Development Model]

- Identify the critical components
- Look up into the library
- Extract components from library (if available)
- Build the component (if not available)
This process model uses the spiral model framework, but dividing the development of components are extracted from the libraries. If the library is a 3rd party library, then the extracted components are called off-the-shelf components.

- If the library is an organisation library, the extracted components are called as fully-partially experienced components.
- The new component functionality are developed from the base line.

After developing new components, place them in the library, so that they may become reusable components for the future projects.

**Estimation Models:**

To estimate the various attributes of the project process/product, there are different empirical models present in the software engineering. Empirical models are derived based on the experience of the past projects without proof. Based on these models, we can calculate the expected estimation of various attributes with software.

The structure of the empirical model is:

\[ E = A + B \cdot (cv)^C \]

- \( E \): effort (man-months)
- \( A, B, C \): empirically derived constants
- \( C \): estimated value (size)
- \( cv \): estimated value (size)

\[ \frac{1 \text{ man-months}}{1 \text{ developer takes 6 months to develop}} \]

\[ \frac{12 \text{ man-months}}{1 \text{ developer takes 12 months to develop}} \]
The sequence of S/W attributes estimation is:

1. SRS
2. Size Estimation
3. Effort Estimation
   - Cost Estimation
   - Schedule Estimation

**Size Estimation:**

- The size of the S/W can be estimated based on the SRS document.
- The size of the S/W can be measured in 2 ways:
  - Direct measurement (LOC)
  - Indirect (Function Point Analysis)

**LOC Estimation (Lines Of Code):**

To calculate the expected size of S/W in terms of LOC, there is a need of dividing the programming skill levels in 3 types:

(i) Experienced → develops Optimistic code (Sopt)
(ii) Average → develops most likely Code (Sm)
(iii) Below Average → develops Pessimistic Code (Spess).

After taking above skill levels, we can use the following formula to calculate the expected size of S/W:
\[ S = \frac{S_{opt} + 4S_{m} + S_{pe}}{6} \]

**Q1.** Slow development Org is planning to develop the \( S \) for geometric applications. During the development process, they have estimated the code as 4600 optimistic, 6900 most likely, 8600 pessimistic, what is the expected size of the \( S \)?

**Ans.** \( S = \frac{4600 + 4 \times 6900 + 8600}{6} = 6800 \) LOC

**Q2.** If the above \( S \) is expected to develop within 12-man-months effort, what is the productivity of 1 developer will use 12 months to develop 6800 Lines of Code?

**Ans.** Productivity of lines of code developed in one month:

\[ \text{Pood} = \frac{6800}{12} = 566.67 \]

**Q3.** If 12-man-months effort project is developed in a 5-months duration, what is the man-power.

**Ans.** \( 1 \times 12 = n \times 5 \)

\( n = 3 \)

3 developers are req.

**Q2.** A company needs to develop the strategy for developing the \( S \) in DSP, in which 2 different programming languages are planned. The LOC developed using lang 1 is estimated...
to be twice the LOC developed using lang 1. The product will have to be maintained for 5 years. Various parameters of the product is given below:-

<table>
<thead>
<tr>
<th>Parameter</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC development man-years</td>
<td>$\frac{LOC = \chi}{10,000}$</td>
<td>$\frac{LOC = 2\chi}{10,000}$</td>
</tr>
<tr>
<td>Dev. Cost/man-year</td>
<td>Rs. 10,00,000</td>
<td>Rs. 7,50,000</td>
</tr>
<tr>
<td>[total = $\chi \times 10$ lakh]</td>
<td></td>
<td>[total = $2\chi \times 15$ lakh]</td>
</tr>
<tr>
<td>Cost of maintenance/year</td>
<td>Rs. 1,00,000</td>
<td>Rs. 50,000</td>
</tr>
<tr>
<td>[total = 5 lakh]</td>
<td></td>
<td>[total = 2.5 lakh]</td>
</tr>
<tr>
<td>Maintenance year</td>
<td>5 years</td>
<td>5 years</td>
</tr>
</tbody>
</table>

Total cost of the product includes cost of the development & maintenance, what is LOC for L1 for which the total cost of project using L1 = total cost of project using L2

\[
\chi \times 10 \text{ lakh} + 5\text{ lakh} = 15\text{ lakh} \times x + 2.5 \text{ lakh}
\]

\[
20 = 2.5 \text{ lakh} - 5\text{ lakh} \times x
\]

\[
x = \frac{4}{2}
\]

\[
LOC = \frac{15,000}{x} \text{ [for L1]}
\]

now $\frac{1}{2}$ man, month effort is req. [for L1] (meaning 6 months req. to develop 5,000 LOC)

\[
\text{Prod} = \frac{5,000}{6} \text{ [i.e., LOC developed in 6 month]}
\]
Function-Point Analysis:

(i) FP Analysis is an indirect measurement used to measure the size of the S/W. To calculate the expected size of the S/W in terms of FP, there is a need of other parameters.

(ii) Five functional parameters & 14 non-functional parameters are req. to calculate the FP of the S/W.

Functional parameters are directly involved in the project:

1. No. of I/Ps: It indicates no. of inflows to the system.
2. No. of O/Ps: It indicates no. of outflows from the system.
3. No. of enquiries: It indicates no. of ways to get the quick info. from the system.
4. No. of files: It indicates the logical connections within the system.
5. No. of external interfaces: It indicates no. of the other modules interfaced with our system.
The five parameter values can be calculated by using:

\[ S = \frac{S_{opt} + 4\times S_{m} + S_{pass}}{6} \]

To calculate the count value of the 5 functional parameters, there is a need for a weighing factor. Weighing factor is derived empirically based on the level of complexity of the project, i.e.

<table>
<thead>
<tr>
<th>Weighing factor:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
</tr>
<tr>
<td>Average</td>
</tr>
<tr>
<td>Complex</td>
</tr>
<tr>
<td>(1)</td>
</tr>
<tr>
<td>(2)</td>
</tr>
<tr>
<td>(3)</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

\[ \text{Count} = \sum W_{ij} \times z_{ij} \]

\[ i = \text{type of the I/O parameter} \]
\[ j = \text{level of complexity} \]
\[ w = \text{weighing factor} \]
\[ z = \text{value of the parameter} \]

* Non-functional requirements are indirectly involved in the project. There is involvement in the project.
<table>
<thead>
<tr>
<th>Involvement</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>If no influence</td>
<td>0</td>
</tr>
<tr>
<td>If Incident</td>
<td>1</td>
</tr>
<tr>
<td>If moderate</td>
<td>2</td>
</tr>
<tr>
<td>If average</td>
<td>3</td>
</tr>
<tr>
<td>If significant</td>
<td>4</td>
</tr>
<tr>
<td>If essential</td>
<td>5</td>
</tr>
</tbody>
</table>

VAF (Value Adjustment Factor) = 0.65 + 0.01 \times \sum_{i=1}^{n} F_i

Now, \[ FP = \text{Count value} \times VAF \]

Q. Consider the sw project with no. of (weighing factor) of i/p's, o/p's, enquires, files & external interfaces at 30(4), 25(5), 20(4), 10(10), 5(7). Assume all complexity AF are constant 4.3. What is the FP for the project?

Ans. Count = 30 \times 4 + 25 \times 5 + 20 \times 4 + 10 \times 10 + 5 \times 7 = 460

\[ VAF = 0.65 + 0.01 \times 3 \times 14 = 1.07 \]

\[ FP = 460 \times 1.07 = 492.2 \]

(b) If the above project requires 2-man years effort, what is the productivity?

Ans. \[
\frac{492.2 \text{ 24 month man-months}}{24 \text{ 1 man-month}} = 492.2 \text{ (prod)}
\]
Q. If in a software project no. of i/p's, no. of o/p's, enquiries, files & external interfaces are 30, 60, 20, 10 & 5 respectively, with complexity & weighing factors as avg., what is the prod. of the project if the effort is 100 man months?

Ans. Count = 120 + 300 + 80 + 100 + 35 = 635

\[ \text{VAF} = 0.06 + 0.01 \times 8 \times 14 \]

\[ = 1.07 \]

\[ \text{FP} = 635 \times 1.07 \]

\[ = 679.45 \]

\[ \text{prod.} = \frac{679.45}{100} = 6.7945 \text{ (FPs developed in 1 month)} \]

Q. A company has to develop a software application. The size of the project can be estimated by using the FP analysis with the following values:

- # of i/p's: 20
- # of o/p's: 12
- # of file inquiries: 16
- # of files: 4
- # of external interfaces: 2

with complexity & weighing factors as avg., what is the productivity of the project, if effort is 2 man-years?

Ans. Count = 4 \times 24.33 + 5 \times 15.67 + 22 \times 4.167 + 7 \times 2.167

\[ = 97.32 + 78.35 + 88 + 15.169 \]

\[ = 320.859 \]

\[ \text{VAF} = 1.07 \]

\[ \text{FP} = 342.94 \]

\[ \text{prod.} = \frac{342.94}{24} = 14.28 \text{ person-month} \]
Halstead size estimation:

1. To measure the various attributes of the program, Halstead defined a diff formula based on the token mechanism.
2. In the LOC count, all the comments & blanks & hash directives are included so it doesn’t give the accurate estimation, a token mechanism is used to estimate the size of the program. In the token mechanism each statement is decomposed into operators & operands. Variables & constants are treated as operands.
3. Function calls & different unary operators & binary operators are treated as operators.
4. The reserved words like return, break, continue, etc. are considered as operators.
5. Diff control structures like if, switch, while, do while are considered as operators.
6. The basic symbols i.e. [ ] ( ) { }, special symbols & termination operators are considered as operators.
7. Goto label, in this Goto is operator, label = operand.
8. In the array accessing array_name[index], array_name = operand & index operand & [3] = operator.
9. Comments & function declarations & hash directives are excluded.
$a = a + (b * c)$

<table>
<thead>
<tr>
<th>Operand</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$+$</td>
</tr>
<tr>
<td>$b$</td>
<td>$*$</td>
</tr>
<tr>
<td>$c$</td>
<td>$\text{()}$</td>
</tr>
</tbody>
</table>

After decomposition the statement into token, the database is required. The database consists of following:

<table>
<thead>
<tr>
<th># of unique operands</th>
<th># of occurrences of the unique operands in the program</th>
<th># of unique operand</th>
<th># of occurrences of the unique operand in the program</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$</td>
<td>1</td>
<td>$a$</td>
<td>2</td>
</tr>
<tr>
<td>$*$</td>
<td>1</td>
<td>$b$</td>
<td>1</td>
</tr>
<tr>
<td>$\text{()}$</td>
<td>1</td>
<td>$c$</td>
<td>1</td>
</tr>
</tbody>
</table>

$Z = n_1$
$Z = N_1$
$Z = n_2$
$Z = N_2$

- After making the database, we can calculate diff attributes of the program.
- The vocabulary of the program: $n = n_1 + n_2$.
- The length of the program: $N = N_1 + N_2$.
- The volume of: $V = N * \log_2{n}$.
- The estimated level of the program: $L = 2n_2$.
- Diff of program is: $D = \frac{1}{L}$.

$\frac{n_1 * n_2}{L}(\text{program level is in the range of } 0 \ldots 1)$
- If $L = 0$, then the program is optimistic.
- If $L = 1$, then the size of the program is low and optimistic code is maintained.

- The estimated length of program: $N = n_1 \log{n_1} + n_2 \log{n_2}$.
- The programming time is calculated as $T = E / B$, $5 \leq B \leq 20$ (normally $B = 18$).
Q1. Consider the following program and calculate different attributes of the program.

```c
int sort(int x[], int n)
{
    int i, j, save, im1;
    if (n < 2) return 1;
    for (i = 2; i <= n; i++)
        im1 = i - 1;
    for (j = 1; j <= im1; j++)
        if (x[i] <= x[j])
        {
            save = x[i];
            x[i] = x[j];
            x[j] = save;
        }
    return 0;
}
```

<table>
<thead>
<tr>
<th>Operator</th>
<th>Occurrences</th>
<th>Operands</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4</td>
<td>x, n</td>
<td>8</td>
</tr>
<tr>
<td>i</td>
<td>7</td>
<td>n</td>
<td>3</td>
</tr>
<tr>
<td>j</td>
<td>5</td>
<td>i, j</td>
<td>3</td>
</tr>
<tr>
<td>=</td>
<td>3</td>
<td>save, im1</td>
<td>3</td>
</tr>
<tr>
<td>&lt;</td>
<td>2</td>
<td>x, j</td>
<td>2</td>
</tr>
<tr>
<td>return</td>
<td>4</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>for</td>
<td>2</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>;</td>
<td>4</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>=</td>
<td>8</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>2</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>-</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td></td>
<td>33</td>
</tr>
</tbody>
</table>
(N) vocabulary = 24 (n1+n2)  
(N) length = 91 (N1+N2)  
\[ N = 14 \log_2 \frac{4}{8^8} + 10 \times \log_2 \frac{10}{8} = 4 + 2 + 3 \times 8 = 86.5 \]  
V = 91 \times 5 \text{ bits}  
= 455 \text{ bits}  
L = 2 \times n_2 = 2 \times 10 = \frac{20}{14 \times 38} = 0.037  
D = \frac{1}{L} = 26.6  
E = \frac{V}{L} = \frac{455 \times 2 \times 6 \times 1 \times 10^3}{0.037} = 12.297  
T = 683.16 \text{ (when } \beta = 18)\]

**Effort Estimation:**

- Based on the size of the S/W, we can estimate the effort required to develop the S/W. Diff. empirical models are used to estimate the effort. The structure of effort is:-

\[ E = A + B(\text{ev})C \]

A, B, & C: - empirical constants  
\( \text{ev} \): estimation value size (KLOC/KPSI/FP)  
A (\text{ev} \ will \ accept \ line \ of \ code \ in \ KLOC)  
\text{Delivery of source ins}^n\]

There are 3 empirical models used to estimate the effort:-

- SEL (S/W Engineering Lab)
- W-F (Walstononald-Felix) [developed by IBM]
- COCOMO (Cost Constructive Model) - Boehm's

**SEL:**

In this model, the effort & duration, both the parameters are calculated as

\[ E = 1.4 (\text{KLOC})^{0.93} \text{ man-months} \]  
\[ D = 4.6 (\text{KLOC})^{0.26} \text{ months} \]
WF Model:

In this model, the effort & duration req. to develop the SW is calculated as:

\[ E = 5.2 (KLOC)^{0.93} \]
\[ D = 4.1 (KLOC)^{0.36} \]

Q. SW development expected to have 8-man years of the effort. Calculate LOC, duration, productivity & avg manning by using S/E & WF models.

Ans.

\[ E = 1.4 (KLOC)^{0.93} \]
\[ 68.57 = (KLOC)^{0.93} \]
\[ 94.26 = (KLOC) \]
\[ \therefore KLOC = 94.26 \]
\[ D = 4.6 \times (94.26)^{0.26} \]
\[ = 21.06 \text{ months} \]

\[ P_{\text{prod}} = \frac{94.26}{E} = 0.9819 \text{ KLOC/month} \]

\[ \text{Avg Manning} = \frac{E}{D} = \frac{96}{15} = 6.4 \] man-months

(how many developers are req. to develop SW of 8-man years into 15 months)
**COCOMO Model:**

- Cocomo model was designed by the Boehm, acc. to Boehm's analysis the projects are classified into 3 types, i.e.
  - Simple
  - Average
  - Complex

Based on the level of complexity, different empirical constants are maintained in 3 different modes:

1. **Organic Mode (2-50 KLOC):**
   - Simple
   - Familiar Application Domain
   - Experienced developers
   - Deadlines are not tight

2. **Semi-Detached:** 1. 50-300 KLOC
   - Medium
   - Deadlines are also medium

3. **Embedded Mode:** 1. 300 KLOC
   - Complex
   - Deadlines are very tight

Cocomo model is divided into 2 types:

1. Basic Cocomo
2. Intermediate Cocomo

Basic Cocomo is used for quick and rough estimation. In this model, accuracy is not possible. The structure of the effort duration estimation is

\[ E = C_1 \times (KLOC)^{P_1} \text{ or } C_1 \times (FP)^{P_1} \]
\[ D = C_2 \times (E)^{P_2} \]

<table>
<thead>
<tr>
<th>Mode</th>
<th>C_1</th>
<th>P_1</th>
<th>C_2</th>
<th>P_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>2.4</td>
<td>1.05</td>
<td>2.5</td>
<td>0.38</td>
</tr>
<tr>
<td>Semi-Detached</td>
<td>3.0</td>
<td>1.12</td>
<td>2.5</td>
<td>0.35</td>
</tr>
<tr>
<td>Embedded</td>
<td>3.6</td>
<td>1.2</td>
<td>2.5</td>
<td>0.32</td>
</tr>
</tbody>
</table>
Q. Suppose that a project was estimated to be 400 K LOC, then calculate the effort & duration by using three modes

**Ans.**

**Organic:**

$$E = 2.4 \times (400)^{1.05} = 1235.31 \text{ man-months}$$

$$D = 2.5 \times (1235.31)^{0.38} = 38.07 \text{ months}$$

**Semi-Detached:**

$$E = 3 \times (400)^{1.12} = 2462.47 \text{ man-months}$$

$$D = 2.5 \times (2462.47)^{0.35} = 38.45 \text{ months}$$

**Embedded**

$$E = 3.6 \times (400)^{1.2} = 4772.81 \text{ man-months}$$

$$D = 2.5 \times (4772.81)^{0.32} = 37.6 \text{ months}$$

**Note:** In the above S/W development, the effort req. to develop the S/W in the embedded mode is 4 times effort req. in organic mode & 2 times in semi-detached mode, but the duration of the 3 modes are is approx. equal.

There is a huge difference in the manpower requirement to develop the S/W on-time & in-budget, there is a need of mode selection i.e. embedded mode.

**Q.** A company needs to develop S/W for one of its newest invention. The S/W is expected to have 40,000 LOC. The company needs to...
determine the effort in person-months needed to develop the SW using the basic COCOMO model. The multiplicative factor for this model is 2.8, while the exponentiation factor is calculated as 1.2, what is estimated effort in person-months.

\[ E = 2.8 \times (40)^{1.2} \]
\[ = 234.25 \text{ person-months.} \]

**Intermediate COCOMO Model:**

- In this model, 15 predictors are introduced as cost drivers to calculate the accurate effort & duration.
- These 15 cost drivers are grouped into 4 types:
  1. Product attributes
  2. Project
  3. Personal
  4. Computer

The cost driver involvement is rated as:
- Low \( \leq 1 \)
- Very Low \( \frac{1}{2} < 1 \)
- Nominal \( = 1 \)
- High \( \geq 2 \)
- Very High \( \geq 1 \)
- Extra High \( > 1 \)

*The cost drivers are ratings are never zero because effort adjustment factor is a multiplication factor of 15 cost drivers.*

The structure of effort & duration estimation is

\[ E = C_1 \times (KLOC \times FP)^{P_1} \times EAF \left( \frac{15}{EAF} \right)^{15} \text{ cost drivers} \]

\[ D = C_2 \times (E)^{P_2} \]
Q. Consider a project that consists of 5 modules with the estimated sizes as:
4K, 2K, 1K, 2K, 3K.
Overall cost & schedule estimate based on the following cost drivers impact:
1.15, 1.15, 0.86, 1.07 & rest others are 1.
What is the expected effort when the multiplicative factor 3.2 & exponentiation factor is 1.05.

Ans. EAF = 844.82 \times 1.05
E = 3.2 \times (12)^{1.05} \times 0.84482 \times 1.05
= 34.34 \text{ man months}
= 52.61 \text{ man months}

Quality Estimation

- Quality is defined as conformance to the explicitly stated requirements.
- Quality is an indirect measurement; it depends on the defects. Defect is an uncovered error after careful evaluation of the project.
- In the quality estimation, we can calculate defect effi density & defect removal efficiency.
- Defect Density = no. of defects x time frame
- Defects (rate) = Opportunities for defects / LOC

(for example, when product is in opm) when it is in opm for

\[
DRE = \frac{E}{E + D} \left[ \frac{\text{Errors} - E \times \text{Total no. of errors}}{D'} - \text{Defects (subset of errors, also called uncovered errors)} \right]
\]

The quality attributes are divided into different types based on the use of
User perspectives:
1. Availability
2. Efficiency
3. Flexibility
4. Integrity
5. Interoperability
6. Reliability
7. Robustness
8. Usability

Developer's perspective:
1. Maintainability
2. Portability
3. Testability
4. Reusability

- In the software engg., the time spent & effort req. keeping sw dbm after the release is very significant & it consumes 40-70% of cost of the entire life cycle.
- To minimize the maintenance effort, invest more effort in the early stage of the sw life cycle.
- To calculate the maintenance effort, Bohem introduced an estimation model.
  (i) ACT (Annual Change)
  (ii) AME (Annual Maintenance Effort)

ACT means the no. of source inst. that undergoes change during a year through adding the instn./deleting the instn./modifying the instn.

\[ ACT = \frac{KLOC_{added} + KLOC_{deleted}}{KLOC_{total}} \]

AME is calculated as:

\[ AME = ACT \times SDE \]

SDE: S/w development effort.
Q1. ACT for a sw system is 15% per year. The development effort is 600-man months. If the life-time of the project is 10 years, what is the total effort of the project?

Ans. In maintenance phase:-

\[
\frac{ACT}{100} = 0.15 \\
AME = 0.15 \times 600 = 90 \text{ man-months (for 1 year)}
\]

for 10 years = 900 man-months.

Now, total effort = 600 + 900 = 1500 man-months.
Functional Testing

1. In this testing, external behaviour is taken into the account to cover the functional errors.
2. In this regard, different test cases are prepared based on the I/P & O/P domain.
3. The test case development process is

\[ \text{I/P} \to \text{O/P} \]

Test Case Development:

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>I/P (Year)</th>
<th>Expected O/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case 1</td>
<td>1</td>
<td>Jan</td>
</tr>
<tr>
<td>Test Case 2</td>
<td>6</td>
<td>June</td>
</tr>
<tr>
<td>Test Case 3</td>
<td>0</td>
<td>Invalid i/p</td>
</tr>
<tr>
<td>Test Case 4</td>
<td>13</td>
<td>Invalid i/p</td>
</tr>
</tbody>
</table>

* The developed test cases are implemented in the system to cover various functional errors.

The implementation process is

Test Case ID | Test Data | Test Result |
-------------|-----------|-------------
Test Case 1  | 1 (actual) | Jan CMP Jan, Success |
Test Case 2  | 0 (actual) | June CMP Invalid i/p, LF failure reported to the development team |
Test Case 4

During the test case implementation, the actual o/p of the system is compared with the expected o/p. When both are same, then the test case is successful, otherwise test case is failure.

The failed test cases are reported back to the development team to modify the program.

In the black box testing, different tests plans are used to cover the functional errors:

1. Equivalence partitioning
2. Boundary value analysis
3. Robustness test
4. Comparison test

Equivalence Partitioning:

In this testing, input domain & output domain is divided into equal parts known as classes. The classes are classified into:

- Valid class
- Invalid class not lower limit.
- Invalid class not upper limit.

According to the above example scenario, input domain is divided into the following classes:

- Class 9 to 12
- Class 0 < 1
- Class 1 > 12
The various test cases are prepared based on the 1/p classes, i.e., test-case 1, 1 to 12, 9, Jan to Dec.

- Test-case 2, \( \frac{1}{9} \) invalid 1/p.
- Test-case 3, \( \frac{1}{129} \) invalid 1/p.

In the test case implementation, test the system with anyone of the 1/p of the class, if the test case is successful with the 1/p, then no need to test the system with other possible 1/p’s of the same class.

**Importance:**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Test Data</th>
<th>Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test-case 1</td>
<td>System Jan</td>
<td>See CMP June</td>
</tr>
<tr>
<td></td>
<td>Success</td>
<td></td>
</tr>
</tbody>
</table>

The advantage of equivalence partitioning testing is that no. of test cases will be minimized.

When the valid 1/p box (class) size is large, then to get the efficient test result there is a need of dividing the class into valid subclasses.

**Boundary Value Analysis**

- Ski test expects identified that most of the errors occur at boundary levels rather than center point of the 1/p domain. ; to cover more errors at boundary levels, boundary value analysis is done.
- In this testing, various test cases are prepared as follows:
  - Lower limit, just above the lower limit, center, just below the upper limit & upper limit.
Suppose that variable \( x \) is bounded between \( a \) and \( b \).

\[
a \leq x \leq b \quad \Rightarrow \quad x = [a, b],
\]

To test variable \( x \), the following test cases are required:

- \( a, a+1, a+b, b-1, b \)

Program contains 2 variables \( x \) and \( y \), both accepting inputs in the range of \([0, 300]\). The various test cases required to test these variables are as follows:

- \([x, 200]\) : To test \( x \), keep \( y \) in the center.
- Keep all other variables (except \( y \)) at the center.

\[
(100, 200), (101, 200), (100, 200), (300, 200), (200, 200), (200, y)
\]

\[
(100, 100), (200, 100), (200, 100), (200, 200), (200, 300)
\]

Note: When the program contains \( n \) variables, then the boundary value analysis yields \( 4n+1 \) test cases.
Robustness Testing

It is an extension of boundary value analysis, it is used to cover more errors at the boundaries rather than center point by conducting the different test case implementation.

Robustness test case development process is:
1. Just below the lower limit, lower limit, just above the lower limit, center, just below the upper limit, upper limit, & just above the upper limit.
2. Suppose that variable \( x \) is bounded between \( a \) & \( b \) values, i.e., \( a \leq x \leq b \), the various test cases seq. to test the variable \( x \) is \( a-1, a, a+1, \frac{a+b}{2}, b-1, b, b+1 \).

\[ \text{e.g. } \text{Prog. contain, } x, y \in [0, 300] \]
\[ \text{Test } x \text{ variable: } (99, 200), (100, 200), (101, 200), (200, 200), (299, 200), (300, 200), (301, 200) \]
\[ \text{Test } y \text{ variable: } (200, 99), (200, 100), (200, 101), (200, 299), (200, 300), (200, 301) \]

\[ \text{Total Test Cases: } 6n+1 \]

\begin{center}
\begin{tikzpicture}
\draw[help lines, color=gray!30, dashed] (0,0) grid (3,3);
\end{tikzpicture}
\end{center}
When the program contains 'n' variables, then the robustness test yields 6n+1 test case.

Comparison Test

- In some of the applications, accuracy is the most important factor, e.g., aircraft simulation.
- In such applications, a predefined OIP set is maintained based on the high component properties and index values.
- After developing the program, the program OIP is always compared with the predefined OIP set to cover the erroneous called as comparison test.

SDLC Testing:

- In the software development life cycle, various test plans are implemented at different levels.
- In SDLC, 3 levels of testing are conducted:
  1. Unit test: Test OIP is performed on each module of the program. In this test, the module correctness is taken into account rather than external behavior of the program. In this testing, various white box test plans are implemented to cover the structural errors.
  2. Integration test: During the integration process, when the new modules are added to the existing system, there is a possibility of
new functional errors because functionality changes as the new modules are added), so some kind of the test is required in the integration process to cover the functional errors. It uses the black box test plans. There are 3 kinds of integration process is present.

(i) TOP DOWN
(ii) Boo BOTTOM-UP
(iii) Sandwich

In TOP down the integration starts from root to leaf. In this process drivers are not required but steps are required. In the BOTTOM-UP, integration starts from leaf node to root node. In this process steps are not req. but drivers case req.

In sandwich, integration starts from the root node & leaf node simultaneously.

Regression Test: It means retest. It is not a development test. It is conducted when the development program is in the operational stage. During the maintainence phase of the product, if any fraction of code is changed in the module, that introduces additional error in the entire system. So there is a need to retest the entire system to cover the errors due to fractional change.

Smoke Test: It is used in the "wrap shrink" applications. When the developer is developing the slow acc to the customer's requirement priority list, some of the modules are added & some are deleted to deploy the build of the system to customer location.
In this process, new functionalities are added so there is possibility of new errors. To cover the functional errors, black box test plans are introduced in this process, known as smoke tests.

(iii) System Test -
During the system test opn, various attributes of the system is tested.

(i) Recovery Test -
It focuses on the backup functionality, i.e. during the operational state, if any unexpected occurs (system crash, power failure, etc) how to recover the data.

(iv) Security Test -
It focuses on the system protection, i.e., what kind of the security is provided (password choice, voice recognition, fingerprint, etc) to protect the system from the unauthorised accesses.

(iii) Stress Test -
It focuses on the load balancing, i.e. how many no. of users can able to use the system at a time.

(iv) Performance Test -
It focuses on the slw reliability, i.e. how long the system is functioning without failure.

Validation Test -
In slw engg. practices, customer evaluation is also an imp. factor, so customer conducts the test opn to cover the errors by applying strange inputs.
Under this case, two kinds of test opn are performed, α & β test.

- α-test is conducted by the customer at the developer site (functional testing). If any errors are identified during the α-test, then the program is immediately modified before deployment because developer is present.

- β-test is conducted by the end user/customer at the customer's location, i.e., β-test is conducted after the deployment. If any errors are identified during β-test, those are reported to the maintenance team.

Support - When the developed product is deployed at the customer's location, support stage is req. to maintain the software from the uncovered errors, platform changes, functional requirement changes, frequent requirement changes.