

Software Quality

Software Reliability

Basic Concepts

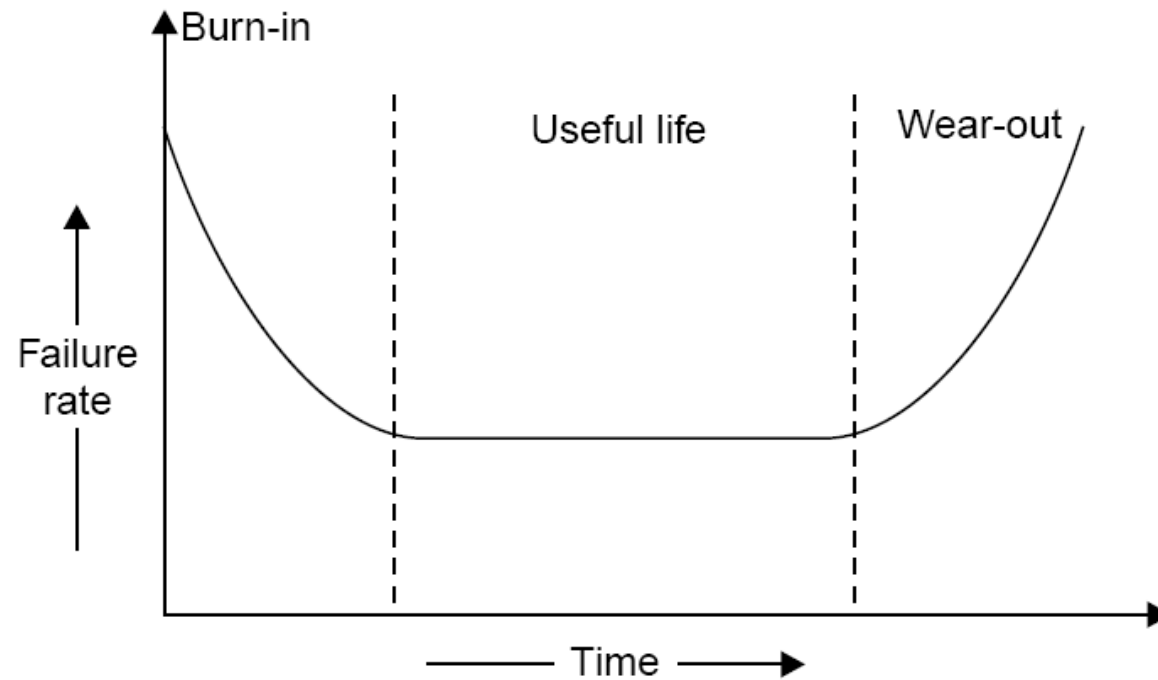
There are three phases in the life of any hardware component i.e., burn-in, useful life & wear-out.

In **burn-in phase**, failure rate is quite high initially, and it starts decreasing gradually as the time progresses.

During **useful life period**, failure rate is approximately constant.

Failure rate increase in **wear-out phase** due to wearing out/aging of components. The best period is useful life period. The shape of this curve is like a “bath tub” and that is why it is known as bath tub curve. The “bath tub curve” is given next.

Software Reliability



Bath tub curve of hardware reliability.

Software Reliability

We do not have wear out phase in software. The expected curve for software is given.

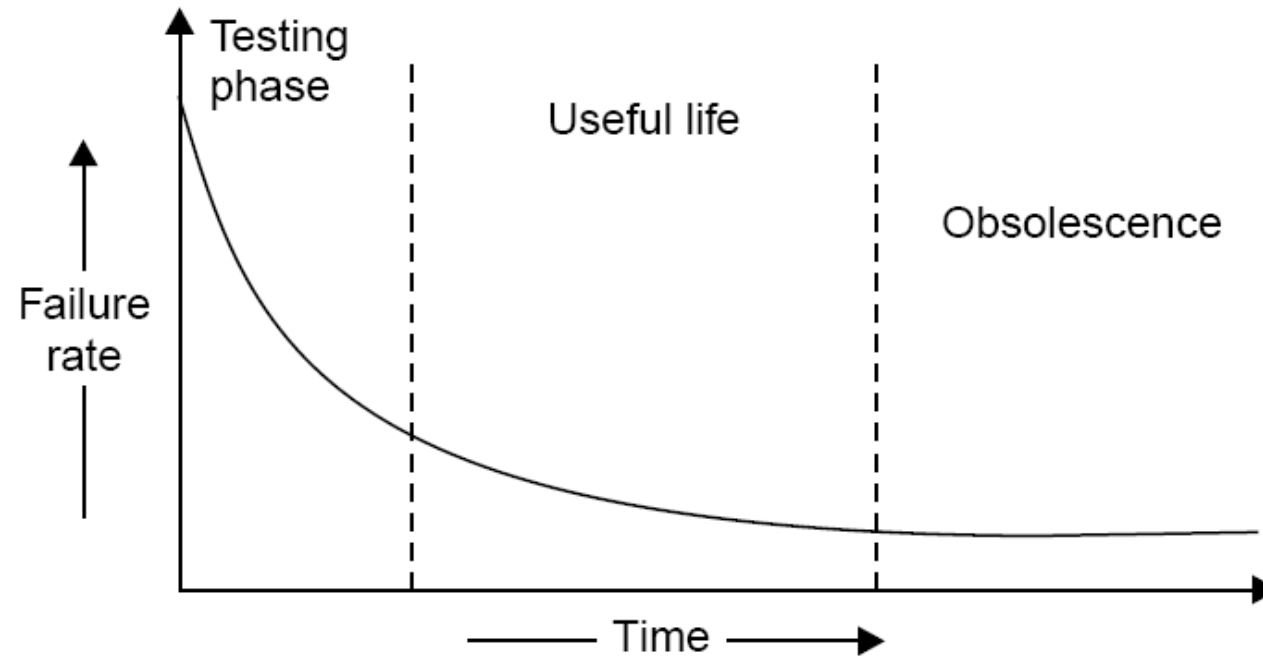


Fig. 7.2: Software reliability curve (failure rate versus time)

Software Reliability

Software may be retired only if it becomes obsolete. Some of contributing factors are given below:

- ✓ change in environment
- ✓ change in infrastructure/technology
- ✓ major change in requirements
- ✓ increase in complexity
- ✓ extremely difficult to maintain
- ✓ deterioration in structure of the code
- ✓ slow execution speed
- ✓ poor graphical user interfaces

Software Reliability

What is Software Reliability?

“Software reliability means operational reliability. Who cares how many bugs are in the program?”

As per IEEE standard: “Software reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time”.

Software Reliability

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the inputs are free of error.

“It is the probability of a failure free operation of a program for a specified time in a specified environment”.

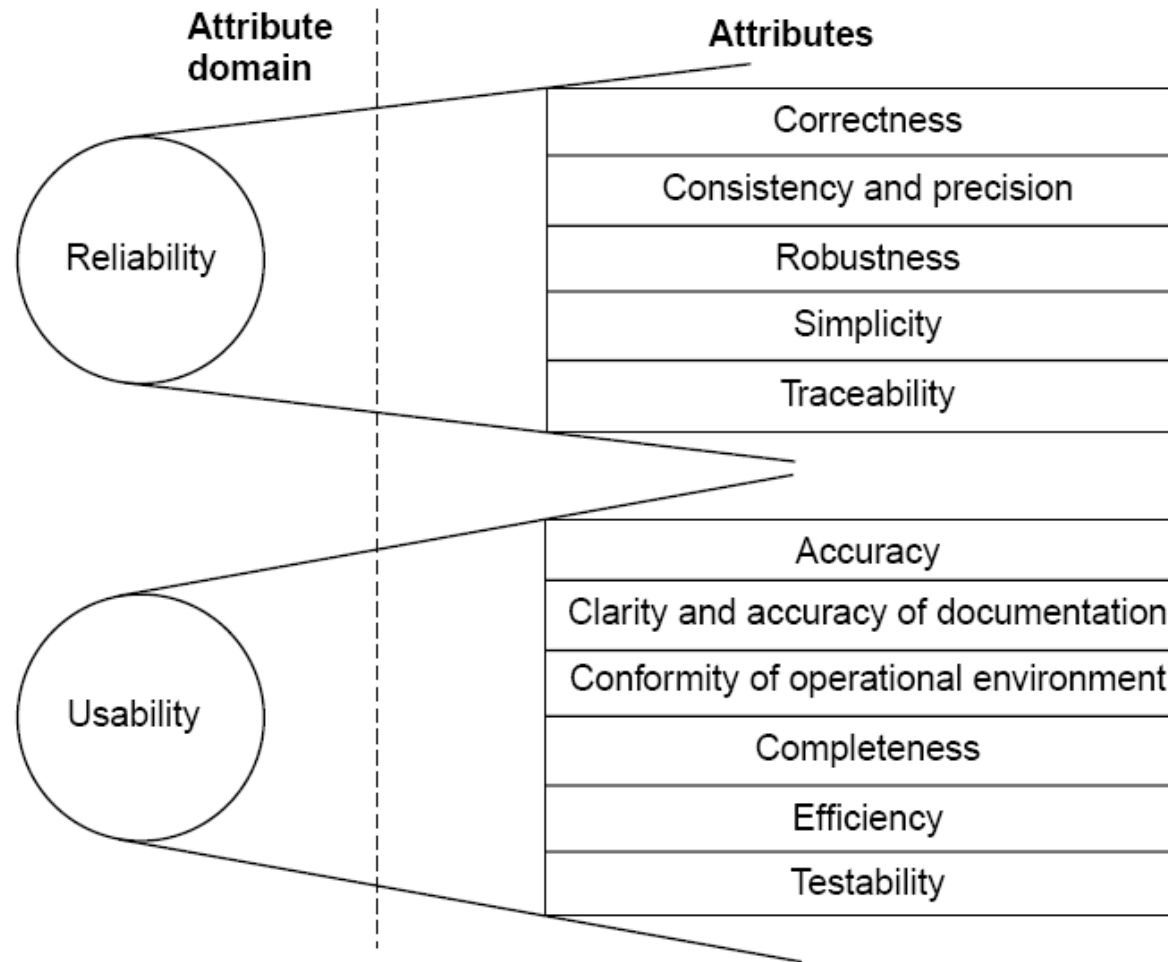
Software Reliability

Software Quality

Different people understand different meanings of quality like:

- ❖ conformance to requirements
- ❖ fitness for the purpose
- ❖ level of satisfaction

Software Reliability



Software Reliability

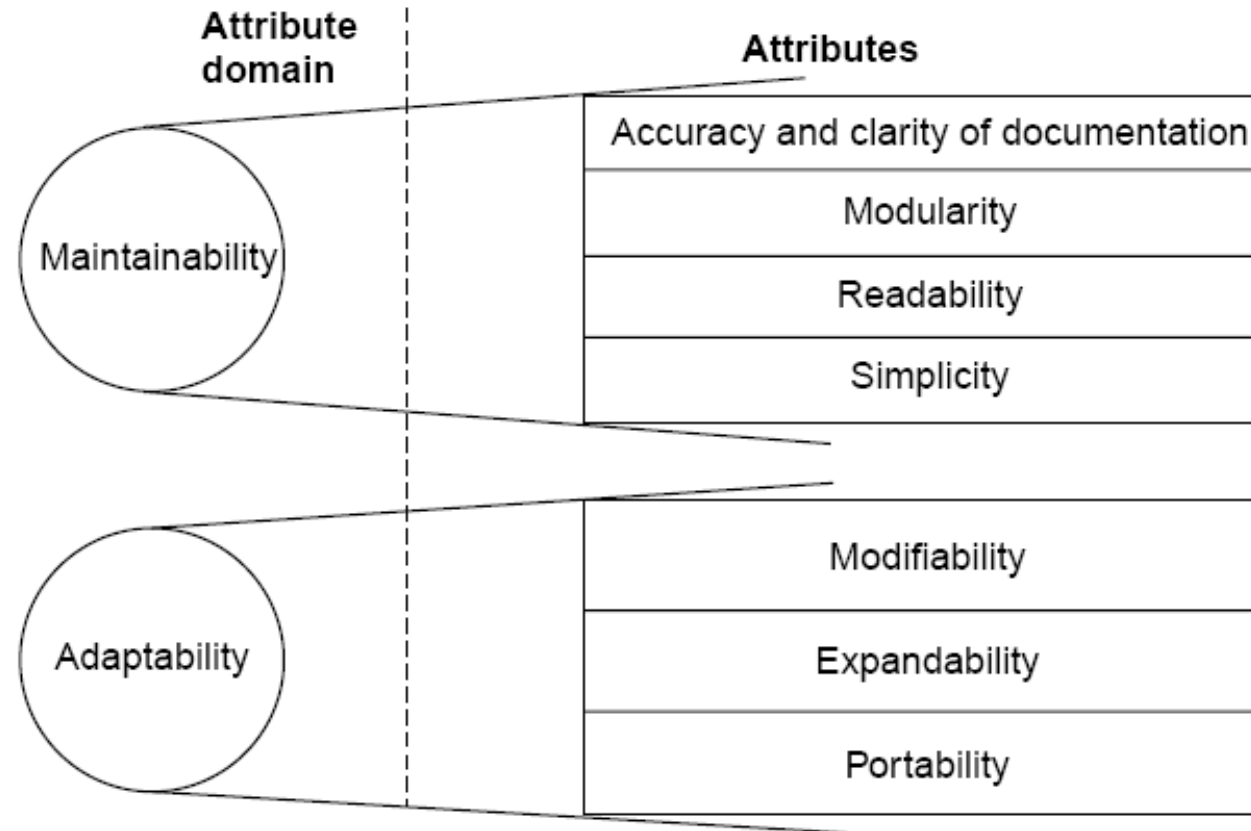


Fig 7.8: Software quality attributes

Software Reliability

1	Reliability	The extent to which a software performs its intended functions without failure.
2	Correctness	The extent to which a software meets its specifications.
3	Consistency & precision	The extent to which a software is consistent and give results with precision.
4	Robustness	The extent to which a software tolerates the unexpected problems.
5	Simplicity	The extent to which a software is simple in its operations.
6	Traceability	The extent to which an error is traceable in order to fix it.
7	Usability	The extent of effort required to learn, operate and understand the functions of the software

(Contd.)...

Software Reliability

8	Accuracy	Meeting specifications with precision.
9	Clarity & Accuracy of documentation	The extent to which documents are clearly & accurately written.
10	Conformity of operational environment	The extent to which a software is in conformity of operational environment.
11	Completeness	The extent to which a software has specified functions.
12	Efficiency	The amount of computing resources and code required by software to perform a function.
13	Testability	The effort required to test a software to ensure that it performs its intended functions.
14	Maintainability	The effort required to locate and fix an error during maintenance phase.

(Contd.)...

Software Reliability

15	Modularity	It is the extent of ease to implement, test, debug and maintain the software.
16	Readability	The extent to which a software is readable in order to understand.
17	Adaptability	The extent to which a software is adaptable to new platforms & technologies.
18	Modifiability	The effort required to modify a software during maintenance phase.
19	Expandability	The extent to which a software is expandable without undesirable side effects.
20	Portability	The effort required to transfer a program from one platform to another platform.

Table 7.4: Software quality attributes

Software Reliability

- McCall Software Quality Model

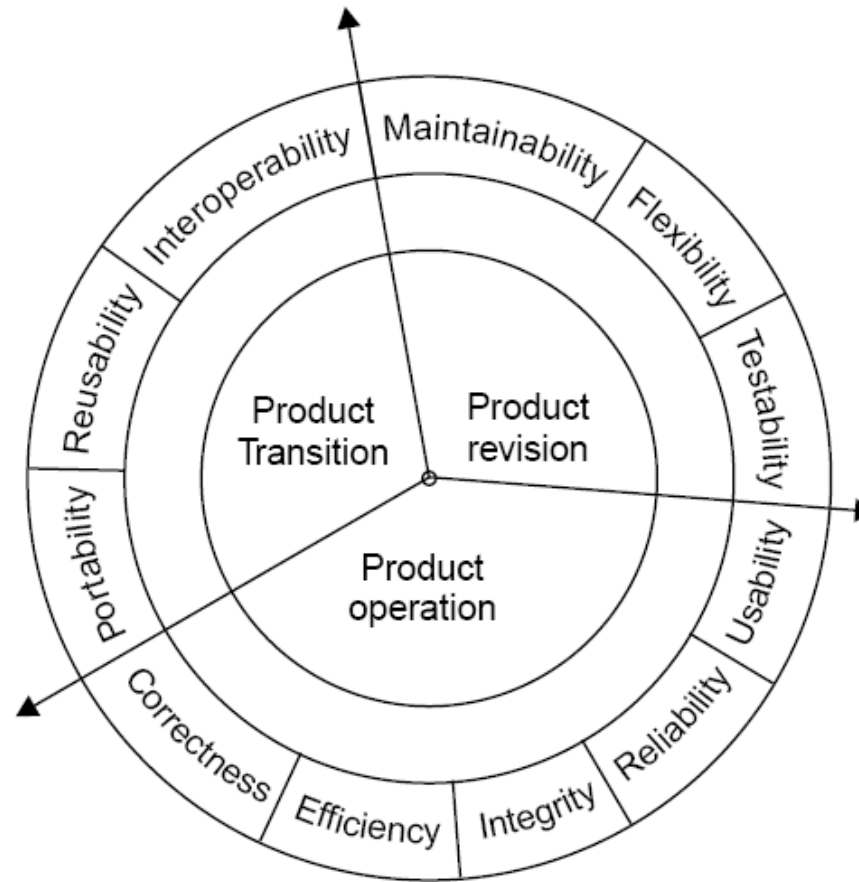


Fig 7.9: Software quality factors

Software Reliability

i. Product Operation

Factors which are related to the operation of a product are combined. The factors are:

- Correctness
- Efficiency
- Integrity
- Reliability
- Usability

These five factors are related to operational performance, convenience, ease of usage and its correctness. These factors play a very significant role in building customer's satisfaction.

Software Reliability

ii. Product Revision

The factors which are required for testing & maintenance are combined and are given below:

- Maintainability
- Flexibility
- Testability

These factors pertain to the testing & maintainability of software. They give us idea about ease of maintenance, flexibility and testing effort. Hence, they are combined under the umbrella of product revision.

Software Reliability

iii. Product Transition

We may have to transfer a product from one platform to an other platform or from one technology to another technology. The factors related to such a transfer are combined and given below:

- Portability
- Reusability
- Interoperability

Software Reliability

Most of the quality factors are explained in table 7.4. The remaining factors are given in table 7.5.

Sr.No.	Quality Factors	Purpose
1	Integrity	The extent to which access to software or data by the unauthorized persons can be controlled.
2	Flexibility	The effort required to modify an operational program.
3	Reusability	The extent to which a program can be reused in other applications.
4	Interoperability	The effort required to couple one system with another.

Table 7.5: Remaining quality factors (other are in table 7.4)

Software Reliability

Sr. No.	Quality Criteria	Usability	Integrity	Efficiency	Correctness	Reliability	Maintainability	Testability	Flexibility	Reusability	Portability	Interoperability
1.	Operability	x										
2.	Training	x										
3.	Communicativeness	x										
4.	I/O volume	x										
5.	I/O rate	x										
6.	Access control		x									
7.	Access Audit		x									
8.	Storage efficiency			x								
9.	Execution Efficiency			x								
10.	Traceability				x							
11.	Completeness				x							
12.	Accuracy					x						
13.	Error tolerance					x						
14.	Consistency				x	x	x					
15.	Simplicity					x	x	x				
16.	Conciseness						x					
17.	Instrumentation							x				
18.	Expandability								x			
19.	Generality								x	x		
20.	Self-descriptiveness						x		x	x	x	
21.	Modularity						x		x	x	x	x
22.	Machine independence									x	x	
23.	S/W system independence									x	x	
24.	Communication commonality											x
25.	Data commonality											x

Table 7.5(a):
Relation
between quality
factors and
quality criteria

Software Reliability

1	Operability	The ease of operation of the software.
2	Training	The ease with which new users can use the system.
3	Communicativeness	The ease with which inputs and outputs can be assimilated.
4	I/O volume	It is related to the I/O volume.
5	I/O rate	It is the indication of I/O rate.
6	Access control	The provisions for control and protection of the software and data.
7	Access audit	The ease with which software and data can be checked for compliance with standards or other requirements.
8	Storage efficiency	The run time storage requirements of the software.
9	Execution efficiency	The run-time efficiency of the software.

(Contd.)...

Software Reliability

10	Traceability	The ability to link software components to requirements.
11	Completeness	The degree to which a full implementation of the required functionality has been achieved.
12	Accuracy	The precision of computations and output.
13	Error tolerance	The degree to which continuity of operation is ensured under adverse conditions.
14	Consistency	The use of uniform design and implementation techniques and notations throughout a project.
15	Simplicity	The ease with which the software can be understood.
16	Conciseness	The compactness of the source code, in terms of lines of code.
17	Instrumentation	The degree to which the software provides for measurements of its use or identification of errors.

(Contd.)...

Software Reliability

18	Expandability	The degree to which storage requirements or software functions can be expanded.
19	Generability	The breadth of the potential application of software components.
20	Self-descriptiveness	The degree to which the documents are self explanatory.
21	Modularity	The provision of highly independent modules.
22	Machine independence	The degree to which software is dependent on its associated hardware.
23	Software system independence	The degree to which software is independent of its environment.
24	Communication commonality	The degree to which standard protocols and interfaces are used.
25	Data commonality	The use of standard data representations.

Table 7.5 (b): Software quality criteria

Software Reliability

ISO 9126

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Software Reliability

Characteristic/ Attribute	Short Description of the Characteristics and the concerns Addressed by Attributes
Functionality	Characteristics relating to achievement of the basic purpose for which the software is being engineered
• Suitability	The presence and appropriateness of a set of functions for specified tasks
• Accuracy	The provision of right or agreed results or effects
• Interoperability	Software's ability to interact with specified systems
• Security	Ability to prevent unauthorized access, whether accidental or deliberate, to program and data.
Reliability	Characteristics relating to capability of software to maintain its level of performance under stated conditions for a stated period of time
• Maturity	Attributes of software that bear on the frequency of failure by faults in the software

(Contd.)...

Software Reliability

• Fault tolerance	Ability to maintain a specified level of performance in cases of software faults or unexpected inputs
• Recoverability	Capability and effort needed to reestablish level of performance and recover affected data after possible failure.
Usability	Characteristics relating to the effort needed for use, and on the individual assessment of such use, by a stated implied set of users.
• Understandability	The effort required for a user to recognize the logical concept and its applicability.
• Learnability	The effort required for a user to learn its application, operation, input and output.
• Operability	The ease of operation and control by users.
Efficiency	Characteristic related to the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

(Contd.)...

Software Reliability

• Time behavior	The speed of response and processing times and throughout rates in performing its function.
• Resource behavior	The amount of resources used and the duration of such use in performing its function.
Maintainability	Characteristics related to the effort needed to make modifications, including corrections, improvements or adaptation of software to changes in environment, requirements and functions specifications.
• Analyzability	The effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified.
• Changeability	The effort needed for modification, fault removal or for environmental change.
• Stability	The risk of unexpected effect of modifications.
• Testability	The effort needed for validating the modified software.

(Contd.)...

Software Reliability

Portability	Characteristics related to the ability to transfer the software from one organization or hardware or software environment to another.
• Adaptability	The opportunity for its adaptation to different specified environments.
• Installability	The effort needed to install the software in a specified environment.
• Conformance	The extent to which it adheres to standards or conventions relating to portability.
• Replaceability	The opportunity and effort of using it in the place of other software in a particular environment.

Table 7.6: Software quality characteristics and attributes – The ISO 9126 view

Software Reliability

- **Capability Maturity Model**

It is a strategy for improving the software process, irrespective of the actual life cycle model used.

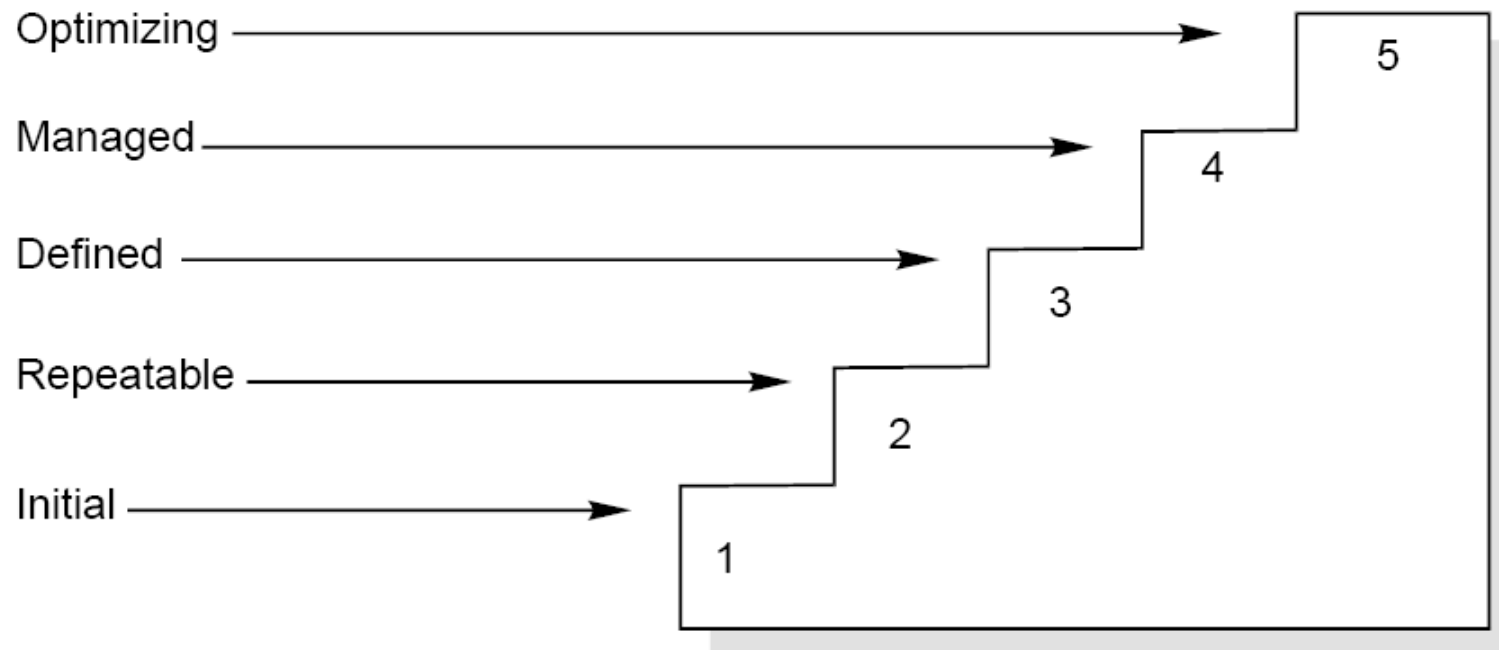
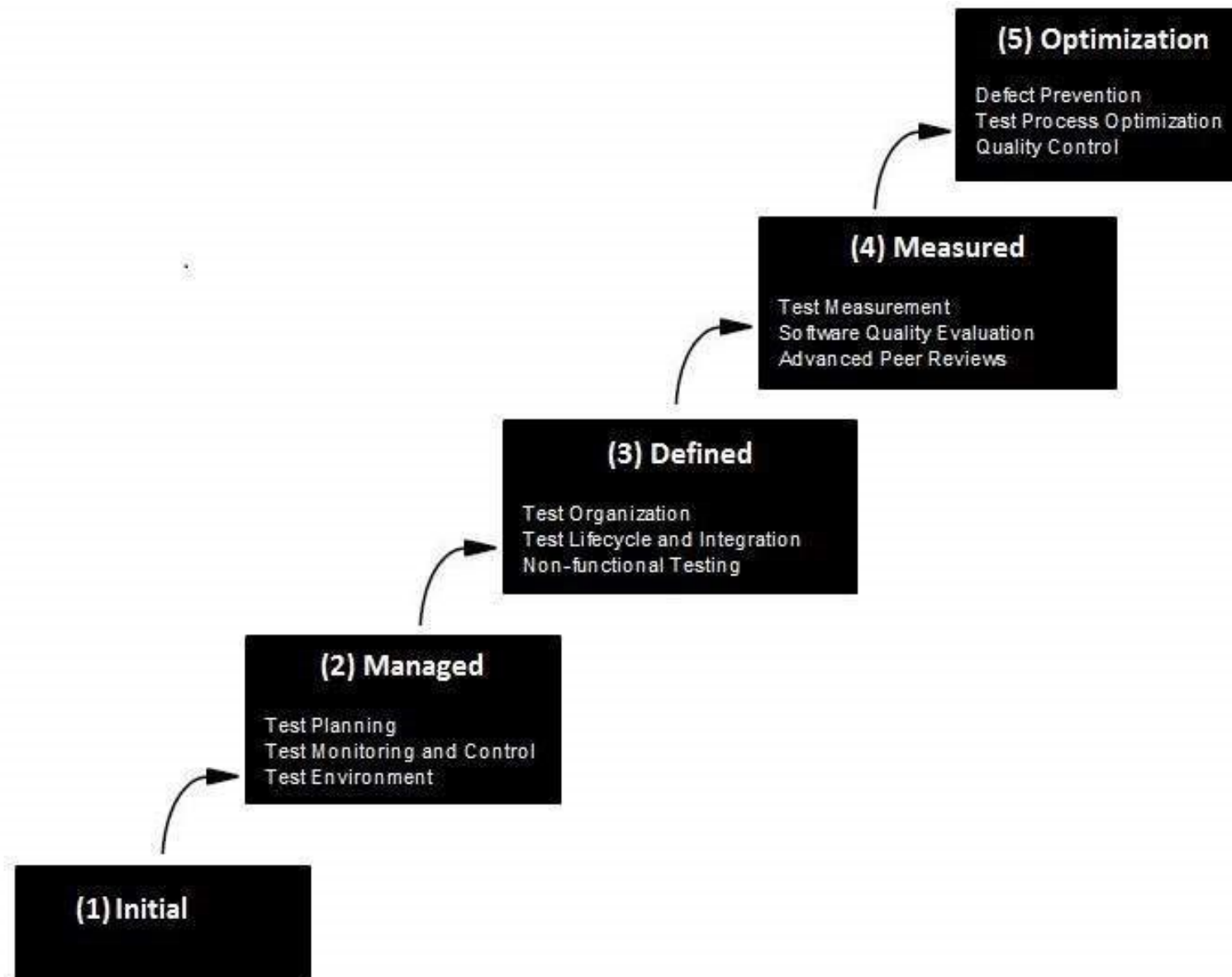


Fig.7.23: Maturity levels of CMM

Software Reliability

Maturity Levels:

- ✓ Initial (Maturity Level 1)
- ✓ Repeatable (Maturity Level 2)
- ✓ Defined (Maturity Level 3)
- ✓ Managed (Maturity Level 4)
- ✓ Optimizing (Maturity Level 5)



Software Reliability

Maternity Level	Characterization
Initial	Adhoc Process
Repeatable	Basic Project Management
Defined	Process Definition
Managed	Process Measurement
Optimizing	Process Control

Fig.7.24: The five levels of CMM

Software Reliability

- **ISO 9000**

The SEI capability maturity model initiative is an attempt to improve software quality by improving the process by which software is developed.

ISO-9000 series of standards is a set of documents dealing with quality systems that can be used for quality assurance purposes. ISO-9000 series is not just software standard. It is a series of five related standards that are applicable to a wide variety of industrial activities, including design/ development, production, installation, and servicing. Within the ISO 9000 Series, standard ISO 9001 for quality system is the standard that is most applicable to software development.

Software Reliability

- Mapping ISO 9001 to the CMM
 1. Management responsibility
 2. Quality system
 3. Contract review
 4. Design control
 5. Document control
 6. Purchasing
 7. Purchaser-supplied product

Software Reliability

- 8. Product identification and traceability
- 9. Process control
- 10. Inspection and testing
- 11. Inspection, measuring and test equipment
- 12. Inspection and test status
- 13. Control of nonconforming product
- 14. Corrective action

Software Reliability

- 15. Handling, storage, packaging and delivery
- 16. Quality records
- 17. Internal quality audits
- 18. Training
- 19. Servicing
- 20. Statistical techniques

Software Reliability

- **Contrasting ISO 9001 and the CMM**

There is a strong correlation between ISO 9001 and the CMM, although some issues in ISO 9001 are not covered in the CMM, and some issues in the CMM are not addressed in ISO 9001.

The biggest difference, however, between these two documents is the emphasis of the CMM on continuous process improvement.

The biggest similarity is that for both the CMM and ISO 9001, the bottom line is **“Say what you do; do what you say”**.

Software Metrics



*“Not everything that can be counted counts,
and not everything that counts can be counted.” –*
Einstein

Software Metrics

Software Metrics: What and Why ?

1. How to measure the size of a software?
2. How much will it cost to develop a software?
3. How many bugs can we expect?
4. When can we stop testing?
5. When can we release the software?

Software Metrics

6. What is the complexity of a module?
7. What is the module strength and coupling?
8. What is the reliability at the time of release?
9. Which test technique is more effective?
10. Are we testing hard or are we testing smart?
11. Do we have a strong program or a weak test suite?

Software Metrics

- ❖ Pressman explained as “A measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of the product or process”.
- ❖ Measurement is the act to determine a measure
- ❖ The metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.

Software Metrics

- **Areas of Applications**

The most established area of software metrics is cost and size estimation techniques.

The prediction of quality levels for software, often in terms of reliability, is another area where software metrics have an important role to play.

The use of software metrics to provide quantitative checks on software design is also a well established area.

Software Metrics

■ Categories of Metrics

- i. **Product metrics:** describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc.
- ii. **Process metrics:** describe the effectiveness and quality of the processes that produce the software product. Examples are:
 - effort required in the process
 - time to produce the product
 - effectiveness of defect removal during development
 - number of defects found during testing

Software Metrics

ii. Project metrics: describe the project characteristics and execution. Examples are :

- number of software developers
- staffing pattern over the life cycle of the software
- cost and schedule
- productivity

Software Metrics

Token Count

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program is defined as:

$$\eta = \eta_1 + \eta_2$$

η : vocabulary of a program

where

η_1 : number of unique operators

η_2 : number of unique operands

Software Metrics

The length of the program in the terms of the total number of tokens used is

$$N = N_1 + N_2$$

where

N : program length

N_1 : total occurrences of operators

N_2 : total occurrences of operands

Software Metrics

- Estimated Program Length

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

$$\hat{N} = 14 \log_2 14 + 10 \log_2 10$$

$$= 53.34 + 33.22 = 86.56$$

Software Metrics

- Counting rules for C language
 1. Comments are not considered.
 2. The identifier and function declarations are not considered.
 3. All the variables and constants are considered operands.
 4. Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.

Software Metrics

5. Local variables with the same name in different functions are counted as unique operands.
6. Functions calls are considered as operators.
7. All looping statements e.g., do {...} while (), while () {...}, for () {...}, all control statements e.g., if () {...}, if () {...} else {...}, etc. are considered as operators.
8. In control construct switch () {case:...}, switch as well as all the case statements are considered as operators.

Software Metrics

9. The reserve words like return, default, continue, break, sizeof, etc., are considered as operators.
10. All the brackets, commas, and terminators are considered as operators.
11. GOTO is counted as an operator and the label is counted as an operand.
12. The unary and binary occurrence of “+” and “-” are dealt separately. Similarly “*” (multiplication operator) are dealt with separately.

Software Metrics

13. In the array variables such as “array-name [index]” “array-name” and “index” are considered as operands and [] is considered as operator.
14. In the structure variables such as “struct-name, member-name” or “struct-name -> member-name”, struct-name, member-name are taken as operands and ‘.’, ‘->’ are taken as operators. Some names of member elements in different structure variables are counted as unique operands.
15. All the hash directive are ignored.

Software Metrics

<i>Language</i>	<i>Language Level λ</i>	<i>Variance σ</i>
PL/1	1.53	0.92
ALGOL	1.21	0.74
FORTRAN	1.14	0.81
CDC Assembly	0.88	0.42
PASCAL	2.54	–
APL	2.42	–
C	0.857	0.445

Table 1: Language levels

Software Metrics

Example- 6.1

Consider the sorting program in Fig. 2 of chapter 4. List out the operators and operands and also calculate the values of software science measures like η, N, V, E, λ etc.

Software Metrics

Solution

The list of operators and operands is given in table 2.

<i>Operators</i>	<i>Occurrences</i>	<i>Operands</i>	<i>Occurrences</i>
int	4	SORT	1
()	5	x	7
,	4	n	3
[]	7	i	8
if	2	j	7
<	2	save	3

(Contd.)...

Software Metrics

;	11	im1	3
for	2	2	2
=	6	1	3
-	1	0	1
< =	2	—	—
+ +	2	—	—
return	2	—	—
{ }	3	—	—
$\eta_1 = 14$	$N_1 = 53$	$\eta_2 = 10$	$N_2 = 38$

Table 2: Operators and operands of sorting program of fig. 2 of chapter 4

Software Metrics

Here $N_1=53$ and $N_2=38$. The program length $N=N_1+N_2=91$

Vocabulary of the program $\eta = \eta_1 + \eta_2 = 14 + 10 = 24$

Volume $V = N \times \log_2 \eta$
 $= 91 \times \log_2 24 = 417 \text{ bits}$

The estimated program length \hat{N} of the program
 $= 14 \log_2 14 + 10 \log_2 10$
 $= 14 * 3.81 + 10 * 3.32$
 $= 53.34 + 33.2 = 86.45$

Software Metrics

Conceptually unique input and output parameters are represented by η_2^*

$\eta_2^* = 3$ {x: array holding the integer to be sorted. This is used both as input and output}.
{N: the size of the array to be sorted}.

The potential volume $V^* = 5 \log_2 5 = 11.6$

Since $L = V^* / V$

Software Metrics

$$= \frac{11.6}{417} = 0.027$$

$$D = I / L$$

$$= \frac{1}{0.027} = 37.03$$

Estimated program level

$$\hat{L} = \frac{2}{\eta_1} \times \frac{\eta_2}{N_2} = \frac{2}{14} \times \frac{10}{38} = 0.038$$

Software Metrics

We may use another formula

$$\hat{V} = V \times \hat{L} = 417 \times 0.038 = 15.67$$

$$\hat{E} = V / \hat{L} = \hat{D} \times V$$

$$= 417 / 0.038 = 10973.68$$

Therefore, 10974 elementary mental discriminations are required to construct the program.

$$T = E / \beta = \frac{10974}{18} = 610 \text{ seconds} = 10 \text{ minutes}$$

This is probably a reasonable time to produce the program, which is very simple

Software Metrics

<code>#include < stdio.h ></code>
<code>#define MAXLINE 100</code>
<code>int getline(char line[],int max);</code>
<code>int strindex(char source[],char search for[]);</code>
<code>char pattern[]="ould";</code>
<code>int main()</code>
<code>{</code>
<code> char line[MAXLINE];</code>
<code> int found = 0;</code>
<code> while(getline(line,MAXLINE)>0)</code>
<code> if(strindex(line, pattern)>=0)</code>
<code> {</code>
<code> printf("%s",line);</code>
<code> found++;</code>
<code> }</code>
<code> return found;</code>
<code>}</code>

Table 3

(Contd.)...

Software Metrics

int getline(char s[],int lim)
{
int c,i=0;
while(--lim > 0 && (c=getchar())!= EOF && c!='\n')
s[i++]=c;
if(c=='\n')
s[i++] = c;
s[i] = '\0';
return i;
}
int strindex(char s[],char t[])
{
int i,j,k;
for(i=0;s[i] !='\0';i++)
{
for(j=i,k=0;t[k] != '\0',s[j] ==t[k];j++,k++);
if(k>0 && t[k] =='\0')
return i;
}
return -1;
}

Table 3

Software Metrics

Example- 6.2

Consider the program shown in Table 3. Calculate the various software science metrics.

Software Metrics

Solution

List of operators and operands are given in Table 4.

<i>Operators</i>	<i>Occurrences</i>	<i>Operands</i>	<i>Occurrences</i>
main ()	1	—	—
—	1	Extern variable pattern	1
for	2	main function line	3
= =	3	found	2
! =	4	getline function s	3
getchar	1	lim	1

Table 4

(Contd.)...

Software Metrics

()	1	<i>c</i>	5
&&	3	<i>i</i>	4
--	1	Strindex function <i>s</i>	2
return	4	<i>t</i>	3
++	6	<i>i</i>	5
printf	1	<i>j</i>	3
>=	1	<i>h</i>	6
strindex	1	Numerical Operands 1	1
If	3	MAXLINE	1
>	3	0	8
getline	1	'\0'	4
while	2	'\n'	2
{ }	5	strings "ould"	1
=	10	—	—
[]	9	—	—
,	6	—	—
;	14	—	—
EOF	1	—	—
$\eta_1 = 24$	$N_1 = 84$	$\eta_2 = 18$	$N_2 = 55$

Table 5

Software Metrics

Program vocabulary $\eta = 42$

Program length $N = N_1 + N_2$
 $= 84 + 55 = 139$

Estimated length $\hat{N} = 24 \log_2 24 + 18 \log_2 18 = 185.115$

% error $= 24.91$

Program volume $V = 749.605$ bits

Estimated program level $= \frac{2}{\eta_1} \times \frac{\eta_2}{N_2}$
 $= \frac{2}{24} \times \frac{18}{55} = 0.02727$

Software Metrics

Minimal volume $V^*=20.4417$

Effort $= V / \hat{L}$

$$= \frac{748.605}{.02727}$$

= 27488.33 elementary mental discriminations.

$$\text{Time } T = E / \beta = \frac{27488.33}{18}$$

= 1527.1295 seconds

= 25.452 minutes

Software Metrics

Cyclomatic Complexity

Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand.

For example, a flow graph shown in in Fig. 21 with entry node 'a' and exit node 'f'.

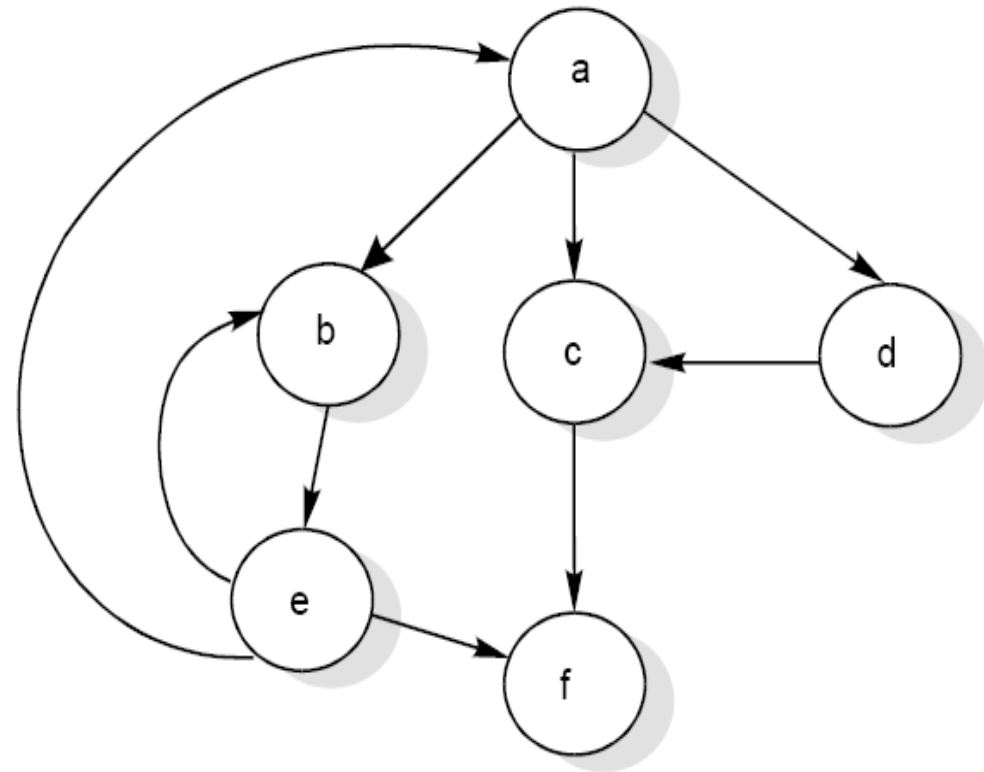


Fig. 21: Flow graph

Software Metrics

The value of cyclomatic complexity can be calculated as :

$$\text{Cyclomatic complexity} = E - N + 2 * P$$

where, E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points

$$V(G) = 9 - 6 + 2 = 5$$

Here $e = 9$, $n = 6$ and $P = 1$

There will be five independent paths for the flow graph illustrated in Fig. 21.

Path 1 : $a c f$

Path 2 : $a b e f$

Path 3 : $a d c f$

Path 4 : $a b e a c f$ or $a b e a b e f$

Path 5 : $a b e b e f$

Software Metrics

Several properties of cyclomatic complexity are stated below:

1. $V(G) \geq 1$
2. $V(G)$ is the maximum number of independent paths in graph G .
3. Inserting & deleting functional statements to G does not affect $V(G)$.
4. G has only one path if and only if $V(G)=1$.
5. Inserting a new row in G increases $V(G)$ by unity.
6. $V(G)$ depends only on the decision structure of G .

Software Metrics

The role of P in the complexity calculation $V(G)=e-n+2P$ is required to be understood correctly. We define a flow graph with unique entry and exit nodes, all nodes reachable from the entry, and exit reachable from all nodes. This definition would result in all flow graphs having only one connected component. One could, however, imagine a main program M and two called subroutines A and B having a flow graph shown in Fig. 22.

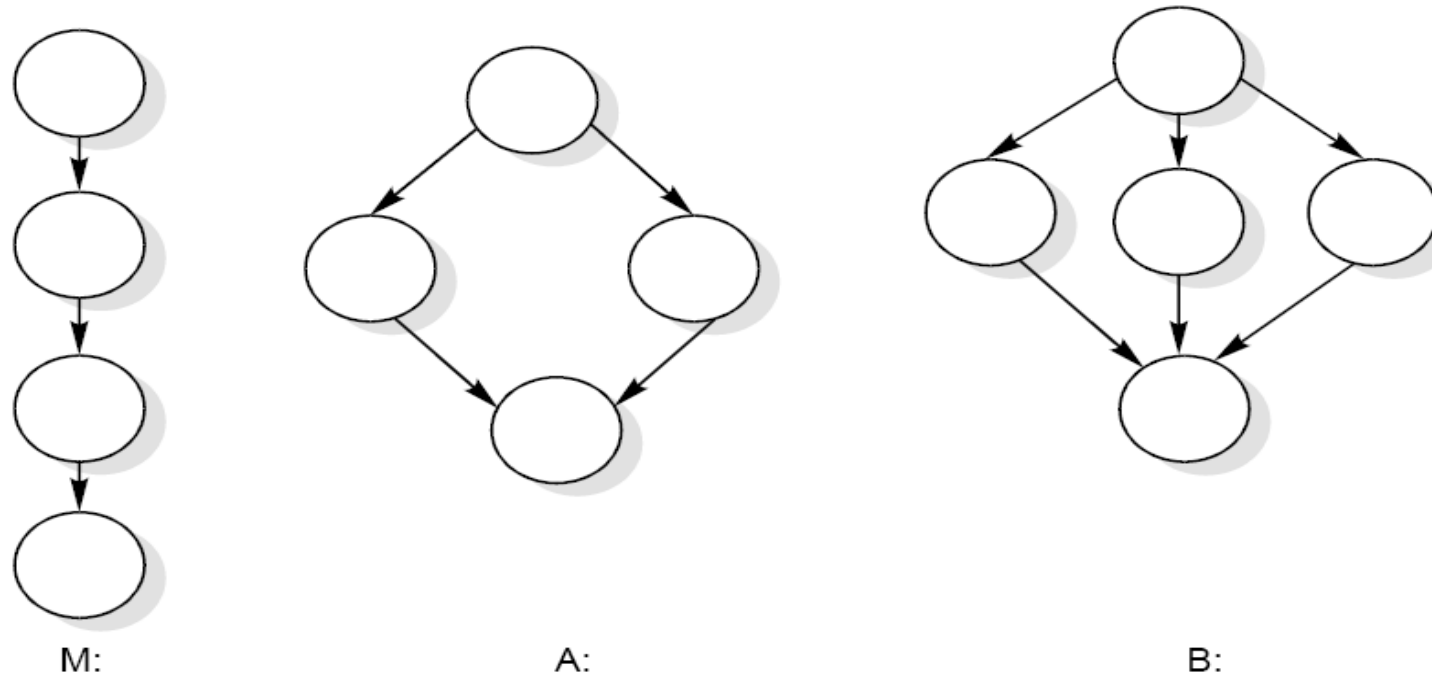


Fig. 22

Software Metrics

Let us denote the total graph above with 3 connected components as

$$\begin{aligned}V(M \cup A \cup B) &= e - n + 2P \\&= 13 - 13 + 2 \cdot 3 \\&= 6\end{aligned}$$

This method with $P \neq 1$ can be used to calculate the complexity of a collection of programs, particularly a hierarchical nest of subroutines.

Software Metrics

Example 8.15

Consider a flow graph given in Fig. 23 and calculate the cyclomatic complexity.

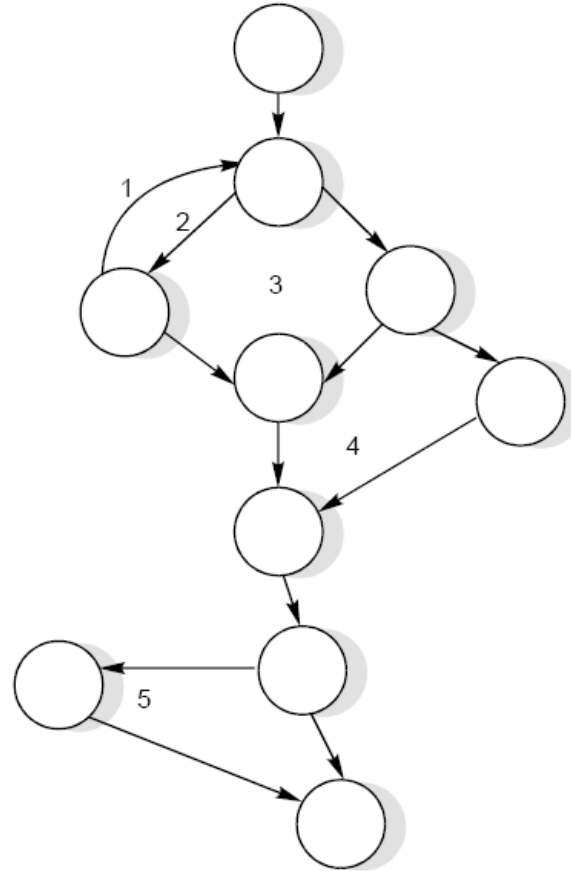


Fig. 23

Software Metrics

Solution

$$\begin{aligned}V(G) &= e - n + 2P \\ &= 13 - 10 + 2 = 5\end{aligned}$$

Therefore, complexity value of a flow graph in Fig. 23 is 5.

COST OF A PROJECT

- The cost in a project is due to:
 - due the requirements for software, hardware and human resources
 - the cost of software development is due to the human resources needed
 - most cost estimates are measured in *person-months (PM)*



COST OF A PROJECT (.)

- the cost of the project depends on the nature and characteristics of the project, at any point, the accuracy of the estimate will depend on the amount of reliable information we have about the final product.



SOFTWARE COST ESTIMATION

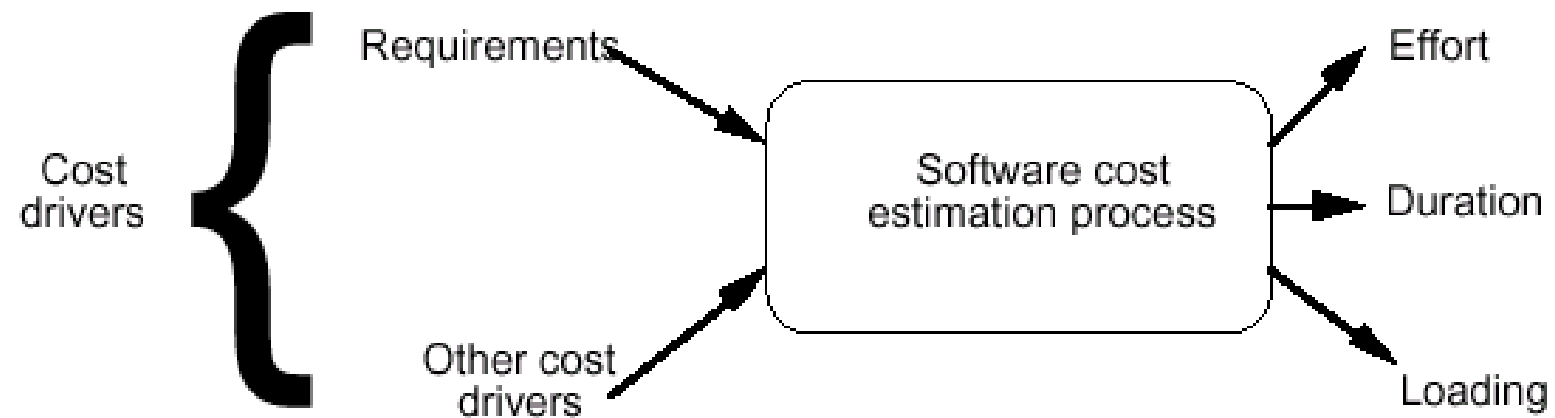


Figure 1. Classical view of software estimation process.



INTRODUCTION TO COCOMO MODELS

- The **CO**structive **CO**st **Model** (COCOMO) is the most widely used software estimation model in the world.
- The COCOMO model predicts the **effort** and **duration** of a project based on inputs relating to the size of the resulting systems and a number of "**cost drives**" that affect productivity.



EFFORT

- Effort Equation

- $PM = C * (KLOC)^n$ (person-months)
 - where **PM** = number of person-month (=152 working hours),
 - **C** = a constant,
 - **KLOC** = thousands of “lines of code” (LOC) and
 - **n** = a constant.



PRODUCTIVITY

- Productivity
 - **(LOC) / (PM)**
 - where **PM** = number of person-month (=152 working hours),
 - **LOC** = " lines of code "



SCHEDULE

- Schedule
 - **$TDEV = C * (PM)^n$** (months)
 - where TDEV = number of months estimated for software development.



AVERAGE STAFFING

- Average Staffing Equation
 - $(PM) / (TDEV) \quad (FSP)$
 - where FSP means Full-time-equivalent Software Personnel.



COCOMO MODELS

- COCOMO is defined in terms of three different models:
 - the **Basic model**,
 - the **Intermediate model**, and
 - the **Detailed model**.
- The more complex models account for more factors that influence software projects, and make more accurate estimates.



THE DEVELOPMENT MODE

- The most important factors contributing to a project's duration and cost is the Development Mode
 - **Organic Mode:** The project is developed in a familiar, stable environment, and the product is similar to previously developed products. The product is relatively small, and requires little innovation.
 - **Semidetached Mode:** The project's characteristics are intermediate between Organic and Embedded.
 - **Embedded Mode:** The project is characterized by tight, inflexible constraints and interface requirements. An embedded mode project will require a great deal of innovation.



MODES

Feature	Organic	Semidetached	Embedded
Organizational understanding of product and objectives	Thorough	Considerable	General
Experience in working with related software systems	Extensive	Considerable	Moderate
Need for software conformance with pre-established requirements	Basic	Considerable	Full
Need for software conformance with external interface specifications	Basic	Considerable	Full

MODES (.)

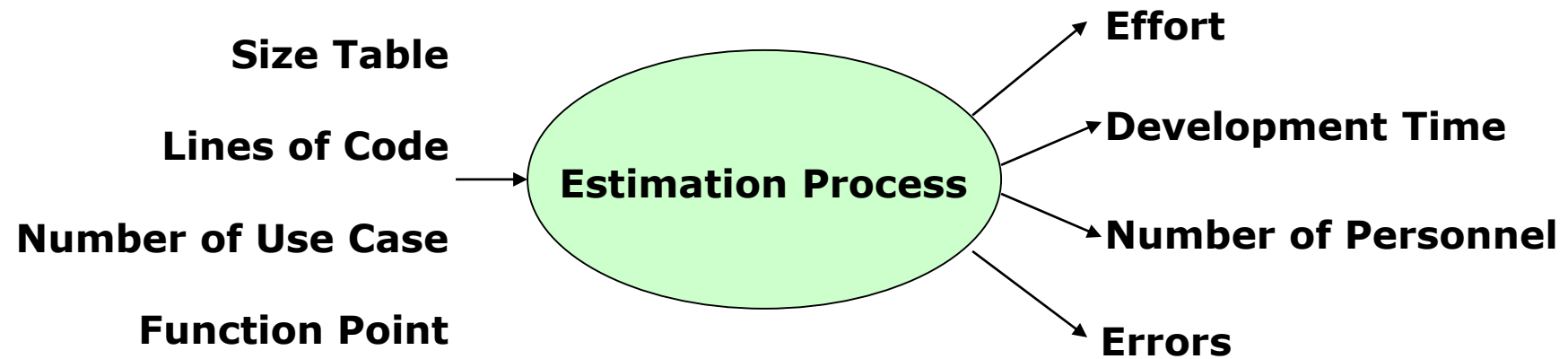
Feature	Organic	Semidetached	Embedded
Concurrent development of associated new hardware and operational procedures	Some	Moderate	Extensive
Need for innovative data processing architectures, algorithms	Minimal	Some	Considerable
Premium on early completion	Low	Medium	High
Product size range	<50 KDSI	<300KDSI	All



COST ESTIMATION PROCESS

Cost=Size of the Project x Productivity

COST ESTIMATION PROCESS



PROJECT SIZE - METRICS

1. **Number of functional requirements**
2. **Cumulative number of functional and non-functional requirements**
3. **Number of Customer Test Cases**
4. **Number of 'typical sized' use cases**
5. **Number of inquiries**
6. **Number of files accessed (external, internal, master)**
7. **Total number of components (subsystems, modules, procedures, routines, classes, methods)**
8. **Total number of interfaces**
9. **Number of System Integration Test Cases**
10. **Number of input and output parameters (summed over each interface)**
11. **Number of Designer Unit Test Cases**
12. **Number of decisions (if, case statements) summed over each routine or method**
13. **Lines of Code, summed over each routine or method**



PROJECT SIZE – METRICS(.)

Availability of Size Estimation Metrics:

	Development Phase	Available Metrics
a	Requirements Gathering	1, 2, 3
b	Requirements Analysis	4, 5
d	High Level Design	6, 7, 8, 9
e	Detailed Design	10, 11, 12
f	Implementation	12, 13



FUNCTION POINTS

- **STEP 1**: measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an ***unadjusted function point count (UFC)***. Counts are made for the following categories
 - • *External inputs* – those items provided by the user that describe distinct application-oriented data (such as file names and menu selections)
 - • *External outputs* – those items provided to the user that generate distinct application-oriented data (such as reports and messages, rather than the individual components of these)
 - • *External inquiries* – interactive inputs requiring a response
 - • *External files* – machine-readable interfaces to other systems
 - • *Internal files* – logical master files in the system





FUNCTION POINTS(..)

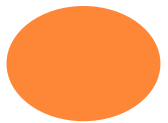
- **STEP 2**: Multiply each number by a weight factor, according to complexity (**simple**, **average** or **complex**) of the parameter, associated with that number. The value is given by a table:

Parameter	simple	average	complex
users inputs	3	4	6
users outputs	4	5	7
users requests	3	4	6
files	7	10	15
external interfaces	5	7	10



ANALYZING THE INFORMATION DOMAIN

<u>measurement parameter</u>	<u>count</u>	<u>weighting factor</u>					
		<u>simple</u>	<u>avg.</u>	<u>complex</u>			
number of user inputs	<input type="text"/>	X	3	4	6	=	<input type="text"/>
number of user outputs	<input type="text"/>	X	4	5	7	=	<input type="text"/>
number of user inquiries	<input type="text"/>	X	3	4	6	=	<input type="text"/>
number of files	<input type="text"/>	X	7	10	15	=	<input type="text"/>
number of ext.interfaces	<input type="text"/>	X	5	7	10	=	<input type="text"/>
count-total							<input type="text"/>
complexity multiplier							<input type="text"/>
function points							<input type="text"/>



FUNCTION POINTS(...)

- **STEP 3**: Calculate the total **UFP** (Unadjusted Function Points)
- **STEP 4**: Calculate the total **TCF** (Technical Complexity Factor) by giving a value between 0 and 5 according to the importance of the following points:



FUNCTION POINTS(...)

○ Technical Complexity Factors:

- 1. Data Communication
- 2. Distributed Data Processing
- 3. Performance Criteria
- 4. Heavily Utilized Hardware
- 5. High Transaction Rates
- 6. Online Data Entry
- 7. Online Updating
- 8. End-user Efficiency
- 9. Complex Computations
- 10. Reusability
- 11. Ease of Installation
- 12. Ease of Operation
- 13. Portability
- 14. Maintainability



FUNCTION POINTS(.....)

- **STEP 5**: Sum the resulting numbers too obtain **DI** (degree of influence)
- **STEP 6**: **TCF** (Technical Complexity Factor) by given by the formula
 - $TCF=0.65+0.01*DI$
- **STEP 6**: Function Points are by given by the formula
 - $FP=UFP*TCF$

$DI = \sum \text{ratings of selected factors}$

$$TCF = 0.65 + 0.01 * \sum_{j=1}^{14} (DI)_j$$

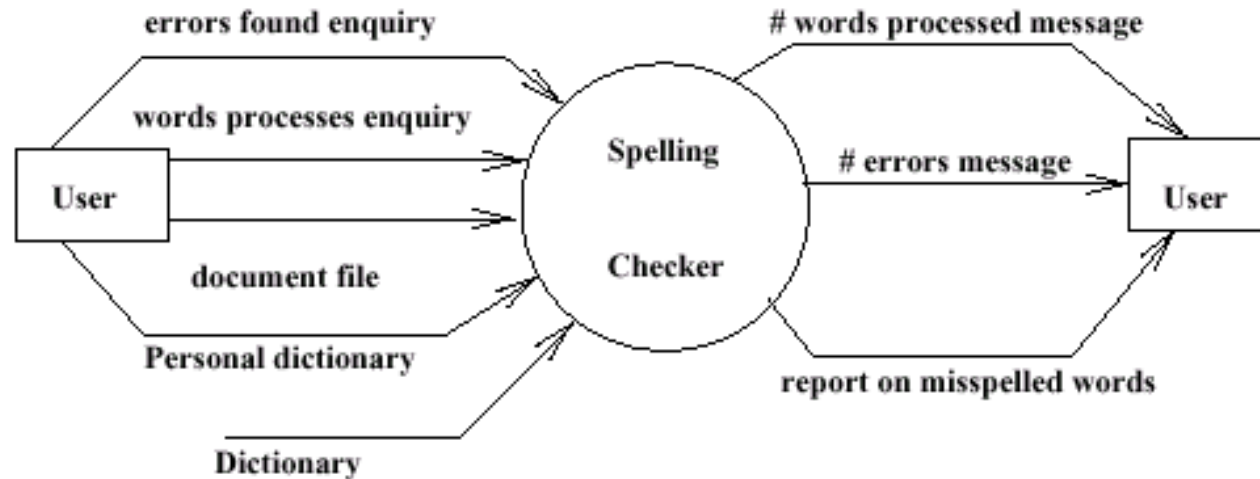
$\text{Min}[TCF] = 0.65; \text{Max}[TCF] = 1.35$



EXAMPLE

Example

The Spell-Checker accepts as input a document file and an optional personal dictionary file. The checker lists all words not contained in either of these files. The user can query the number of words processed and the number of spelling errors found at any stage during processing.



EXAMPLE (.)

- 2 users inputs: document file name, personal dictionary name (average)
- 3 users outputs: fault report, word count, misspelled error count (average)
- 2 users requests: #treated words?, #found errors? (average)
- 1 internal file: dictionary (average)
- 2 external files: document file, personal dictionary (av).

$$UFP = 4 \times 2 + 5 \times 3 + 4 \times 2 + 10 \times 1 + 7 \times 2 = 55$$



EXAMPLE (..)

○ Technical Complexity Factors:

- | | | |
|-------|-----------------------------|---|
| ○ 1. | Data Communication | 3 |
| ○ 2. | Distributed Data Processing | 0 |
| ○ 3. | Performance Criteria | 4 |
| ○ 4. | Heavily Utilized Hardware | 0 |
| ○ 5. | High Transaction Rates | 3 |
| ○ 6. | Online Data Entry | 3 |
| ○ 7. | Online Updating | 3 |
| ○ 8. | End-user Efficiency | 3 |
| ○ 9. | Complex Computations | 0 |
| ○ 10. | Reusability | 3 |
| ○ 11. | Ease of Installation | 3 |
| ○ 12. | Ease of Operation | 5 |
| ○ 13. | Portability | 3 |
| ○ 14. | Maintainability | 3 |

- **DI =30 (Degree of Influence)**



EXAMPLE (...)

- Function Points

- $FP = UFP * (0.65 + 0.01 * DI) = 55 * (0.65 + 0.01 * 30) = 52.25$
- That means the is **FP=52.25**



EXERCISE: FUNCTION POINTS

- Compute the function point value for a project with the following information domain characteristics:
 - Number of user inputs: 32
 - Number of user outputs: 60
 - Number of user enquiries: 24
 - Number of files: 8
 - Number of external interfaces: 2
 - Assume that weights are average and external complexity adjustment values are not important.
- Answer:



RELATION BETWEEN LOC AND FP

- Relationship:

- $$LOC = \text{Language Factor} * FP$$

- where

- LOC (Lines of Code)
 - FP (Function Points)

Relation between LOC and FPs

Language	LOC/FP
assembly	320
C	128
Cobol	105
Fortan	105
Pascal	90
Ada	70
OO languages	30
4GL languages	20



RELATION BETWEEN LOC AND FP(.)

Assuming LOC's per FP for:

Java = 53,

C++ = 64

$$\mathbf{aKLOC = FP * LOC_per_FP / 1000}$$

It means for the SpellChecker example: (Java)

$$\mathbf{LOC=52.25*53=2769.25 \text{ LOC or } 2.76 \text{ KLOC}}$$



EFFORT COMPUTATION

- The **Basic COCOMO model** computes effort as a function of program size. The Basic COCOMO equation is:

$$E = aKLOC^b$$

- Effort for three modes of Basic COCOMO.

Mode	a	b
<i>Organic</i>	2.4	1.05
<i>Semi-detached</i>	3.0	1.12
<i>Embedded</i>	3.6	1.20



EXAMPLE

Mode	Effort Formula
Organic	$E = 2.4 * (S^{1.05})$
Semidetached	$E = 3.0 * (S^{1.12})$
Embedded	$E = 3.6 * (S^{1.20})$

Size = 200 KLOC

Effort = a * Size^b

Organic — $E = 2.4 * (200^{1.05}) = 626$ staff-months

Semidetached — $E = 3.0 * (200^{1.12}) = 1133$ staff-months

Embedded — $E = 3.6 * (200^{1.20}) = 2077$ staff-months



EFFORT COMPUTATION

- The **intermediate COCOMO model** computes effort as a function of program size and a set of cost drivers. The Intermediate COCOMO equation is:
 - $E = aKLOC^b * EAF$
- Effort for three modes of intermediate COCOMO.

Mode	a	b
<i>Organic</i>	3.2	1.05
<i>Semi-detached</i>	3.0	1.12
<i>Embedded</i>	2.8	1.20



EFFORT COMPUTATION(.)

○ Effort Adjustment Factor

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Required Reliability	.75	.88	1.00	1.15	1.40	1.40
Database Size	.94	.94	1.00	1.08	1.16	1.16
Product Complexity	.70	.85	1.00	1.15	1.30	1.65
Execution Time Constraint	1.00	1.00	1.00	1.11	1.30	1.66
Main Storage Constraint	1.00	1.00	1.00	1.06	1.21	1.56
Virtual Machine Volatility	.87	.87	1.00	1.15	1.30	1.30
Comp Turn Around Time	.87	.87	1.00	1.07	1.15	1.15
Analyst Capability	1.46	1.19	1.00	.86	.71	.71
Application Experience	1.29	1.13	1.00	.91	.82	.82
Programmers Capability	1.42	1.17	1.00	.86	.70	.70
Virtual machine Experience	1.21	1.10	1.00	.90	.90	.90
Language Experience	1.14	1.07	1.00	.95	.95	.95
Modern Prog Practices	1.24	1.10	1.00	.91	.82	.82
SW Tools	1.24	1.10	1.00	.91	.83	.83
Required Dev Schedule	1.23	1.08	1.00	1.04	1.10	1,10



EFFORT COMPUTATION (..)

Total EAF = Product of the selected factors

Adjusted value of Effort: Adjusted Person Months:

$$\mathbf{APM = (Total\ EAF) * PM}$$



EXAMPLE

	Organic	Semidetached	Embedded	Mode	Effort Formula
a	3.2	3.0	2.8	Organic	$E = 3.2 * (S^{1.05}) * C$
b	1.05	1.12	1.20	Semidetached	$E = 3.0 * (S^{1.12}) * C$
				Embedded	$E = 2.8 * (S^{1.20}) * C$

e.g. Size = 200 KLOC

Effort = $a * \text{Size}^b * C$

Cost drivers:

Low reliability .88

High product complexity 1.15

Low application experience 1.13

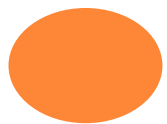
High programming language experience .95

$$C = .88 * 1.15 * 1.13 * .95 = 1.086$$

Organic — $E = 3.2 * (200^{1.05}) * 1.086 = 906$ staff-months

Semidetached — $E = 3.0 * (200^{1.12}) * 1.086 = 1231$ staff-months

Embedded — $E = 2.8 * (200^{1.20}) * 1.086 = 1755$ staff-months



SOFTWARE DEVELOPMENT TIME

○ Development Time Equation Parameter Table:


Parameter	Organic	Semi-detached	Embedded
<i>C</i>	2.5	2.5	2.5
<i>D</i>	0.38	0.35	0.32

Development Time, **$TDEV = C * (APM * D)$**

Number of Personnel, **$NP = APM / TDEV$**



DISTRIBUTION OF EFFORT

- A development process typically consists of the following stages:
 - - **Requirements Analysis**
 - - **Design (High Level + Detailed)**
 - - **Implementation & Coding**
 - - **Testing (Unit + Integration)**
- 

DISTRIBUTION OF EFFORT (.)

The following table gives the recommended **percentage distribution of Effort (APM)** and **TDEV** for these stages:

Percentage Distribution of Effort and Time Table:

	Req Analysis	Design, HLD + DD	Implementation	Testing	
Effort	23%	29%	22%	21%	100%
TDEV	39%	25%	15%	21%	100%

