Software Reliability

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 in Fig.7.1.

Software Reliability

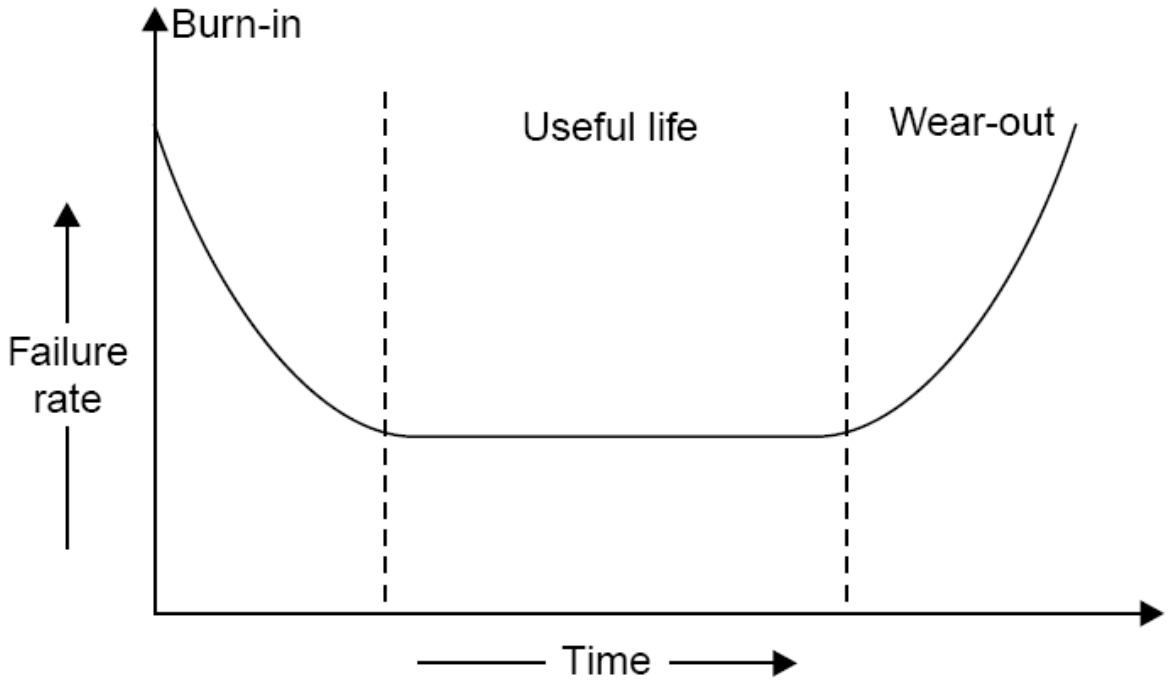


Fig. 7.1: Bath tub curve of hardware reliability.

Software Reliability

We do not have wear out phase in software. The expected curve for software is given in fig. 7.2.

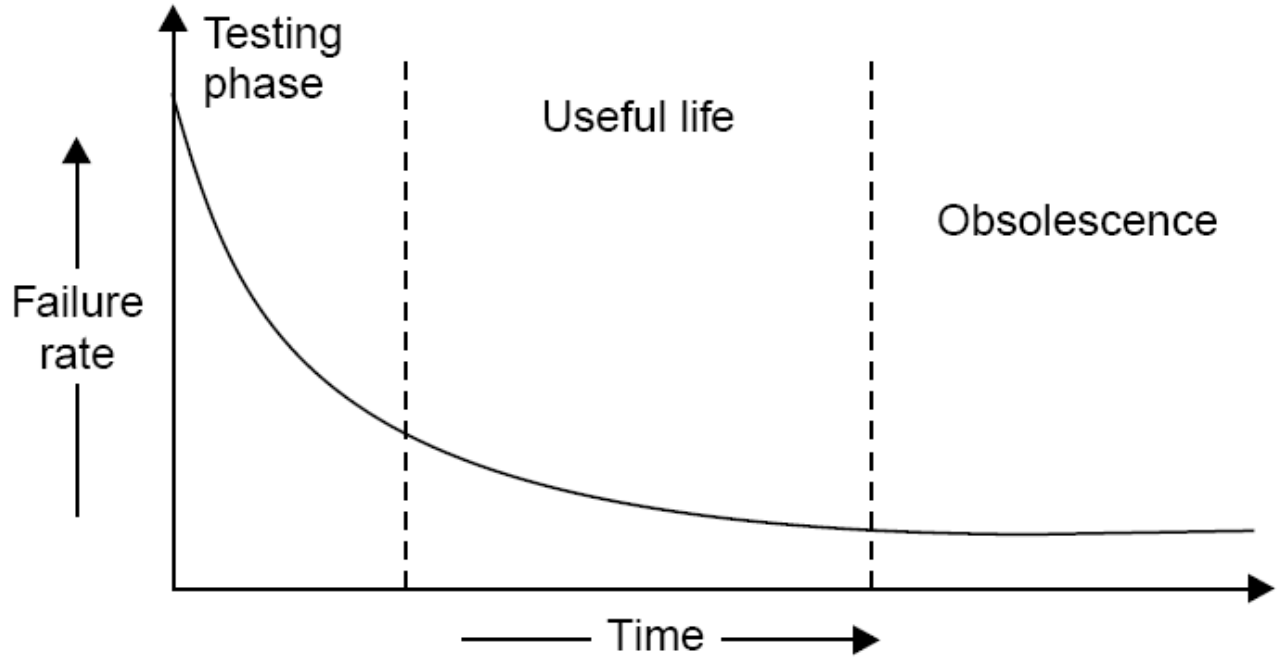


Fig. 7.2: Software reliability curve (failure rate versus time)

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

- Failures and Faults

A fault is the defect in the program that, when executed under particular conditions, causes a failure.

The execution time for a program is the time that is actually spent by a processor in executing the instructions of that program. The second kind of time is calendar time. It is the familiar time that we normally experience.

Software Reliability

There are four general ways of characterising failure occurrences in time:

1. time of failure,
2. time interval between failures,
3. cumulative failure experienced up to a given time,
4. failures experienced in a time interval.

Software Reliability

Failure Number	Failure Time (sec)	Failure interval (sec)
1	8	8
2	18	10
3	25	7
4	36	11
5	45	9
6	57	12
7	71	14
8	86	15
9	104	18
10	124	20
11	143	19
12	169	26
13	197	28
14	222	25
15	250	28

Table 7.1: Time based failure specification

Software Reliability

Time (sec)	Cumulative Failures	Failure in interval (30 sec)
30	3	3
60	6	3
90	8	2
120	9	1
150	11	2
180	12	1
210	13	1
240	14	1

Table 7.2: Failure based failure specification

Software Reliability

Value of random variable (failures in time period)	Probability	
	Elapsed time t_A = 1 hr	Elapsed time t_B = 5 hr
0	0.10	0.01
1	0.18	0.02
2	0.22	0.03
3	0.16	0.04
4	0.11	0.05
5	0.08	0.07
6	0.05	0.09
7	0.04	0.12
8	0.03	0.16
9	0.02	0.13

Table 7.3: Probability distribution at times t_A and t_B

Software Reliability

Value of random variable (failures in time period)	Probability	
	Elapsed time t_A = 1 hr	Elapsed time t_B = 5 hr
10	0.01	0.10
11	0	0.07
12	0	0.05
13	0	0.03
14	0	0.02
15	0	0.01
Mean failures	3.04	7.77

Table 7.3: Probability distribution at times t_A and t_B

Software Reliability

A random process whose probability distribution varies with time to time is called non-homogeneous. Most failure processes during test fit this situation. Fig. 7.3 illustrates the mean value and the related failure intensity functions at time t_A and t_B. Note that the mean failures experienced increases from 3.04 to 7.77 between these two points, while the failure intensity decreases.

Failure behavior is affected by two principal factors:

- ✓ the number of faults in the software being executed.
- ✓ the execution environment or the operational profile of execution.

Software Reliability

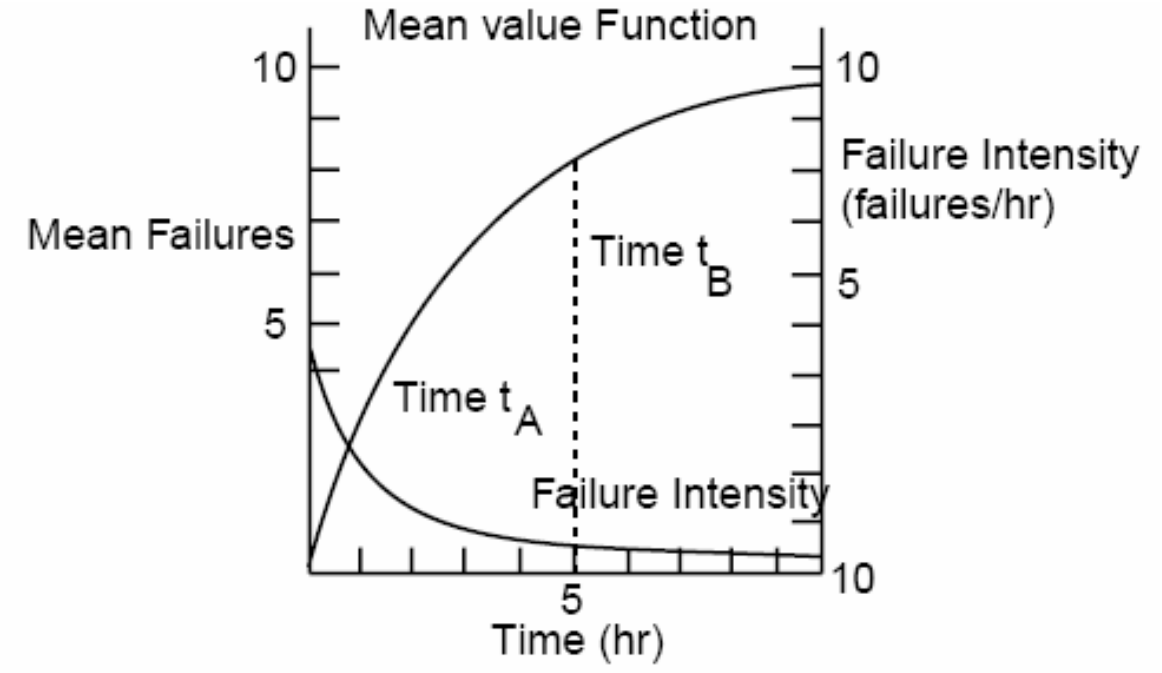


Fig. 7.3: Mean Value & failure intensity functions.

Software Reliability

Environment

The environment is described by the operational profile. The proportion of runs of various types may vary, depending on the functional environment. Examples of a run type might be:

1. a particular transaction in an airline reservation system or a business data processing system,
2. a specific cycle in a closed loop control system (for example, in a chemical process industry),
3. a particular service performed by an operating system for a user.

Software Reliability

The run types required of the program by the environment can be viewed as being selected randomly. Thus, we define the operational profile as the set of run types that the program can execute along with possibilities with which they will occur. In fig. 7.4, we show two of many possible input states A and B, with their probabilities of occurrence.

The part of the operational profile for just these two states is shown in fig. 7.5. A realistic operational profile is illustrated in fig.7.6.

Software Reliability

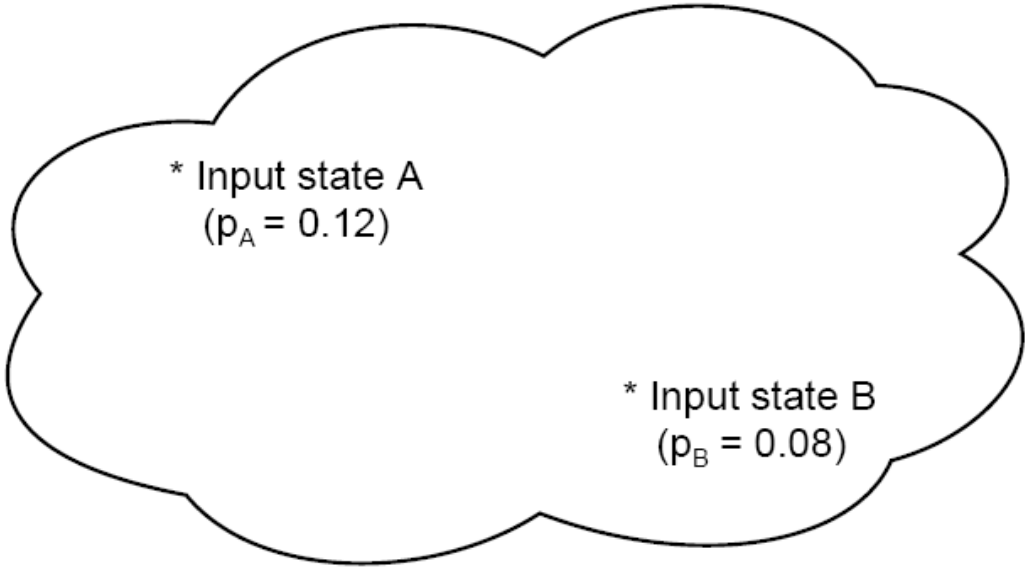


Fig. 7.4: Input Space

Software Reliability

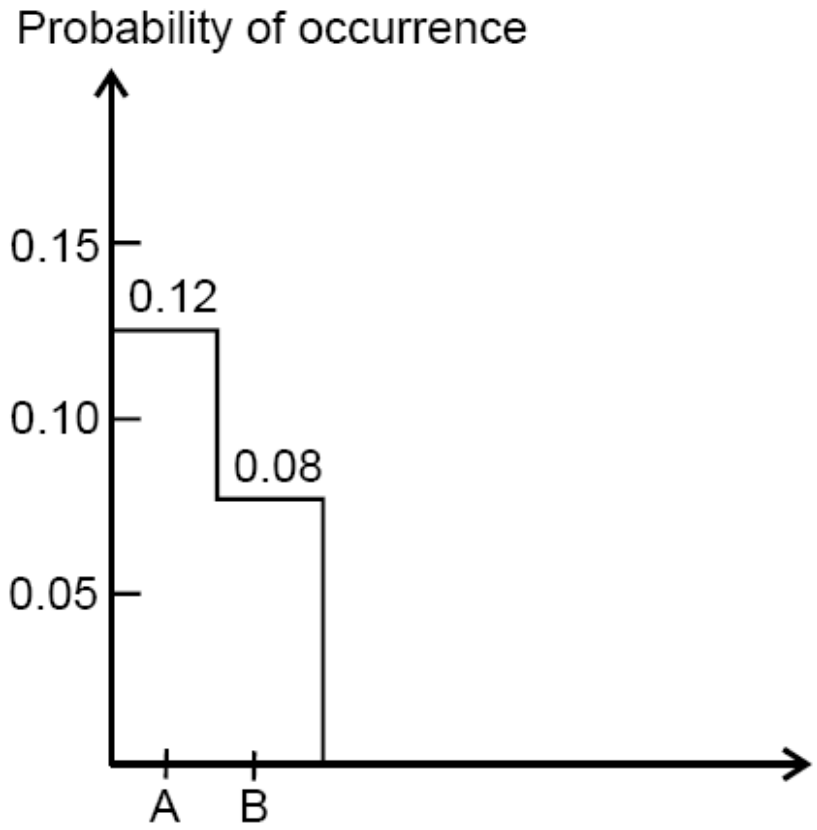


Fig. 7.5: Portion of operational profile

Software Reliability

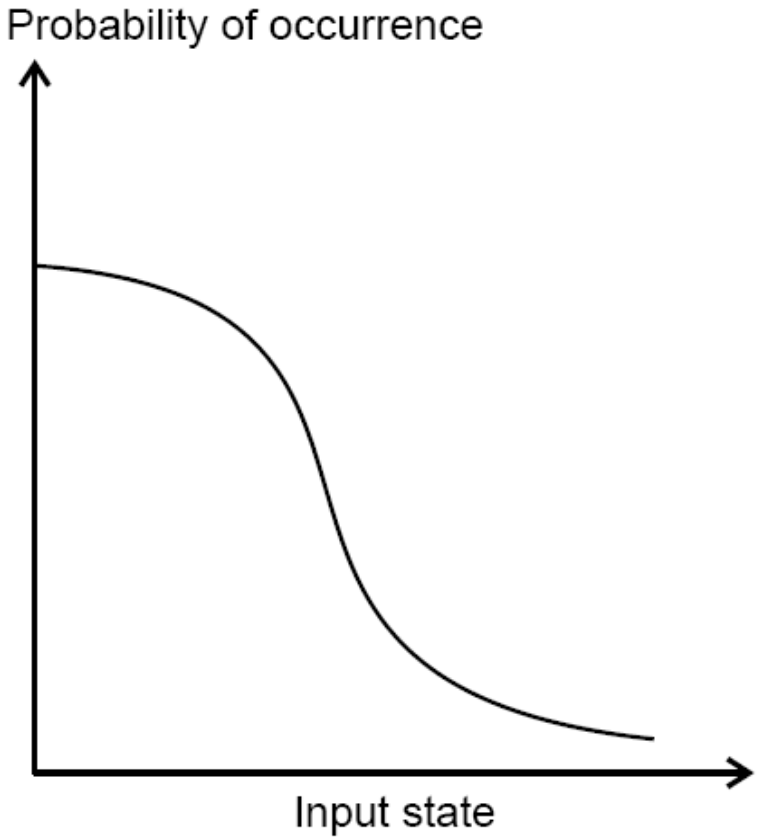


Fig. 7.6: Operational profile

Software Reliability

Fig.7.7 shows how failure intensity and reliability typically vary during a test period, as faults are removed.

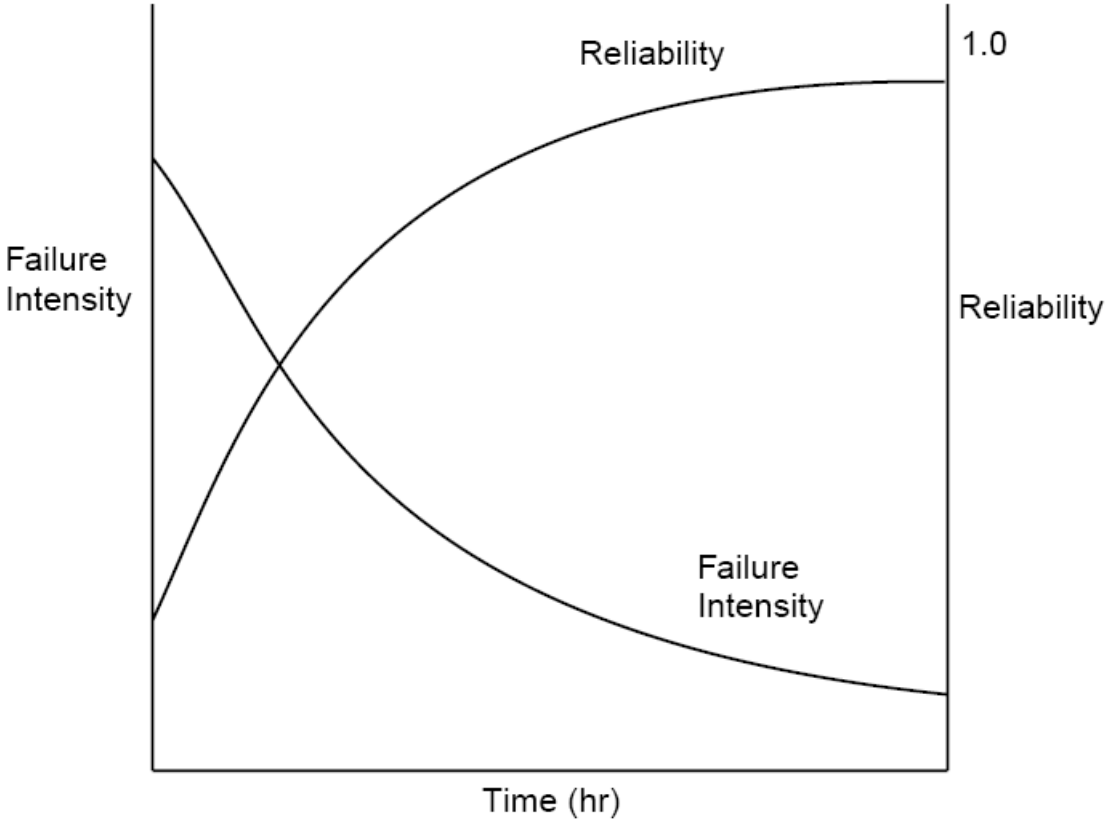


Fig. 7.7: Reliability and failure intensity

Software Reliability

Uses of Reliability Studies

There are at least four other ways in which software reliability measures can be of great value to the software engineer, manager or user.

1. you can use software reliability measures to evaluate software engineering technology quantitatively.

2. software reliability measures offer you the possibility of evaluating development status during the test phases of a project.

Software Reliability

3. one can use software reliability measures to monitor the operational performance of software and to control new features added and design changes made to the software.

4. a quantitative understanding of software quality and the various factors influencing it and affected by it enriches into the software product and the software development process.

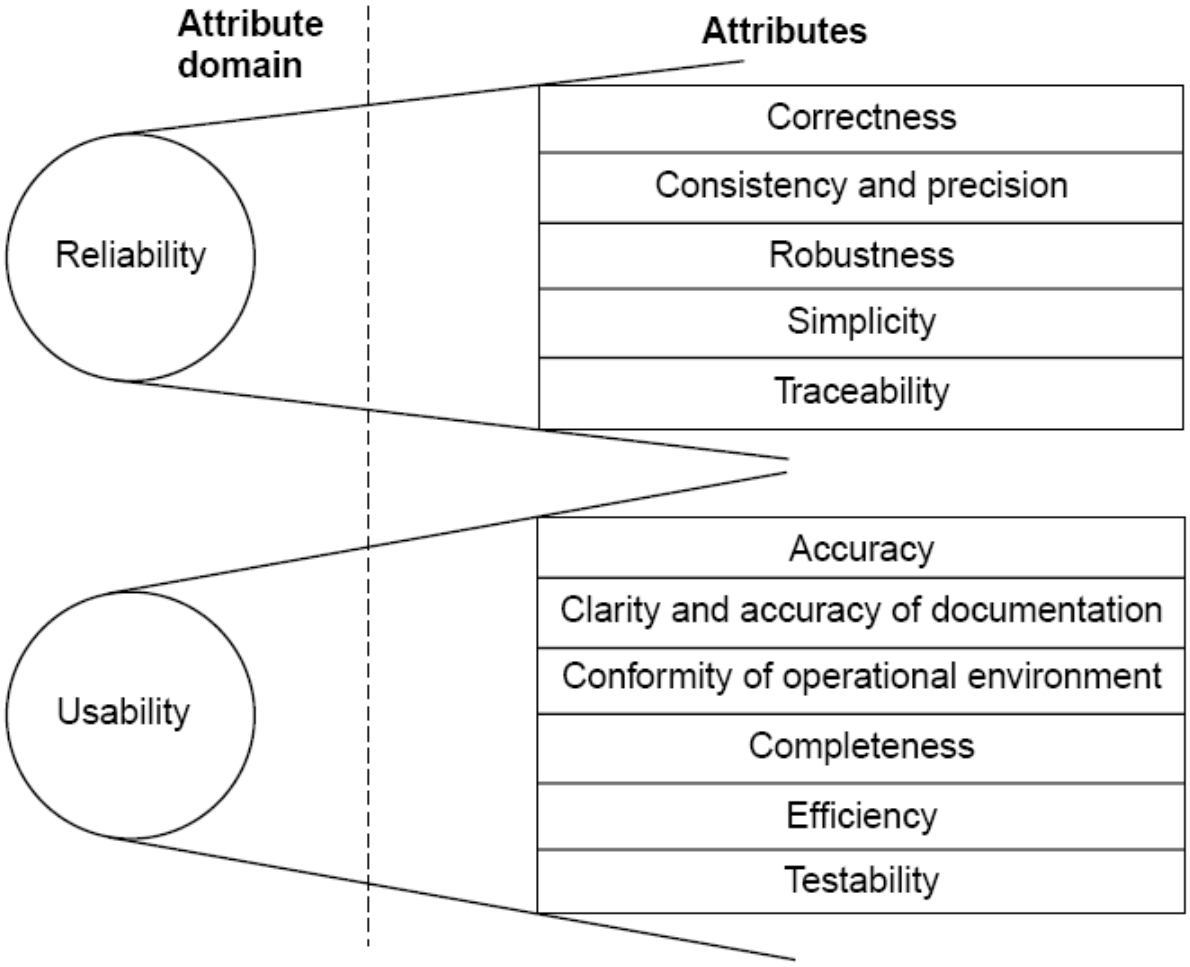
Software Reliability

Software Quality

Different people understand different meanings of quality like:

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

Software Reliability



Software Reliability

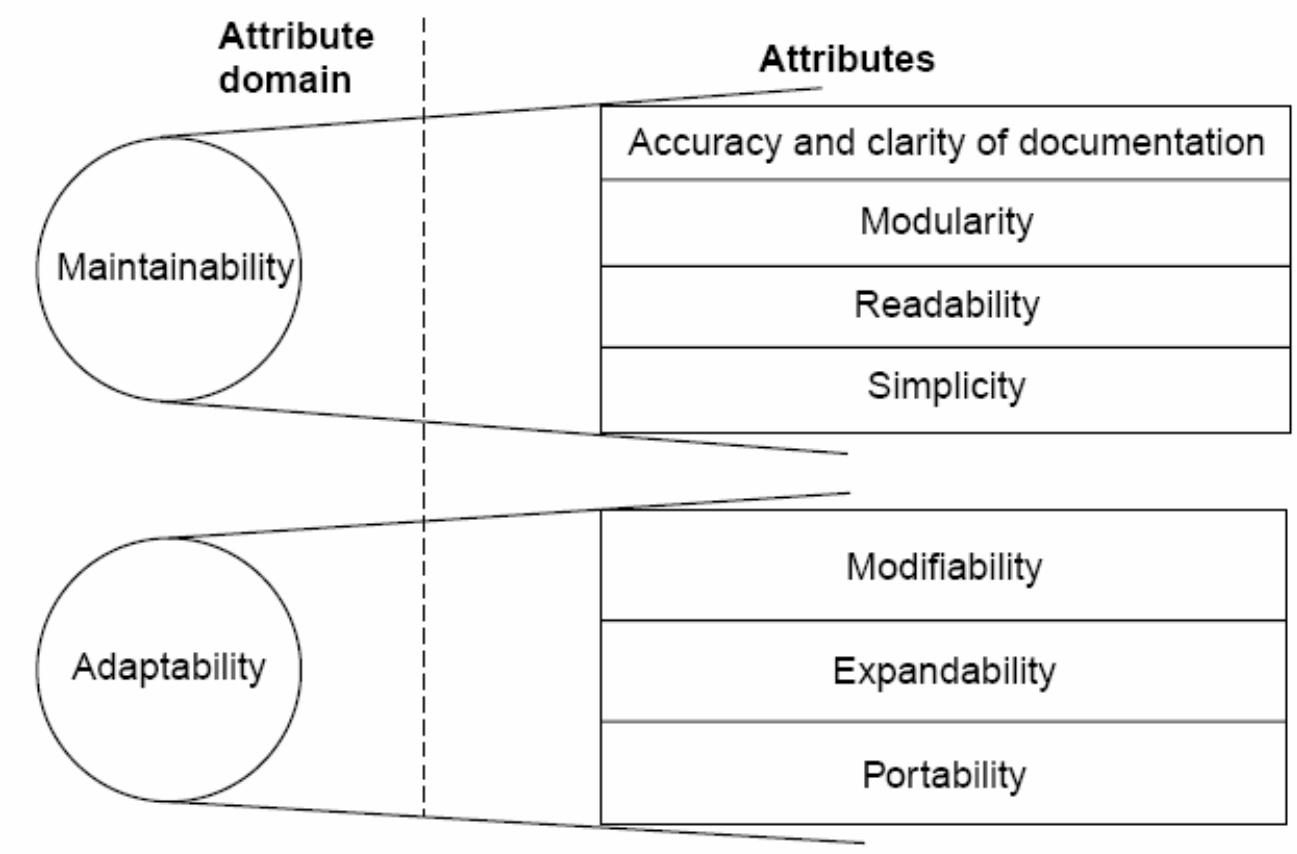


Fig 7.8: Software quality attributes

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

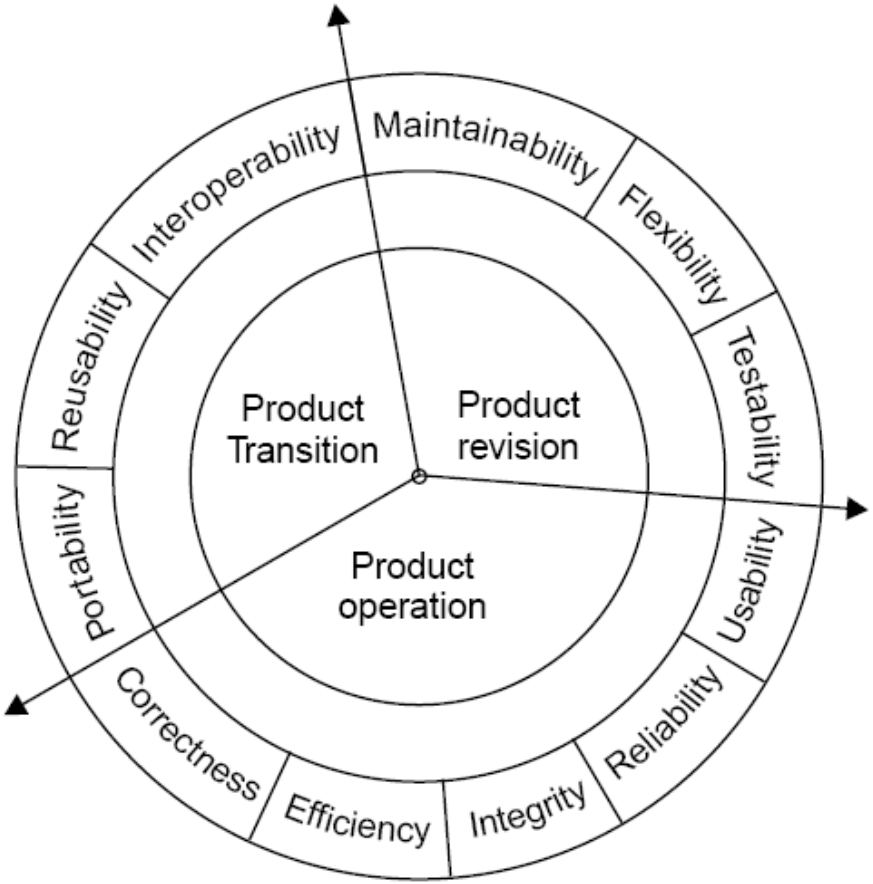


Fig 7.9: Software quality factors

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)

Quality criteria

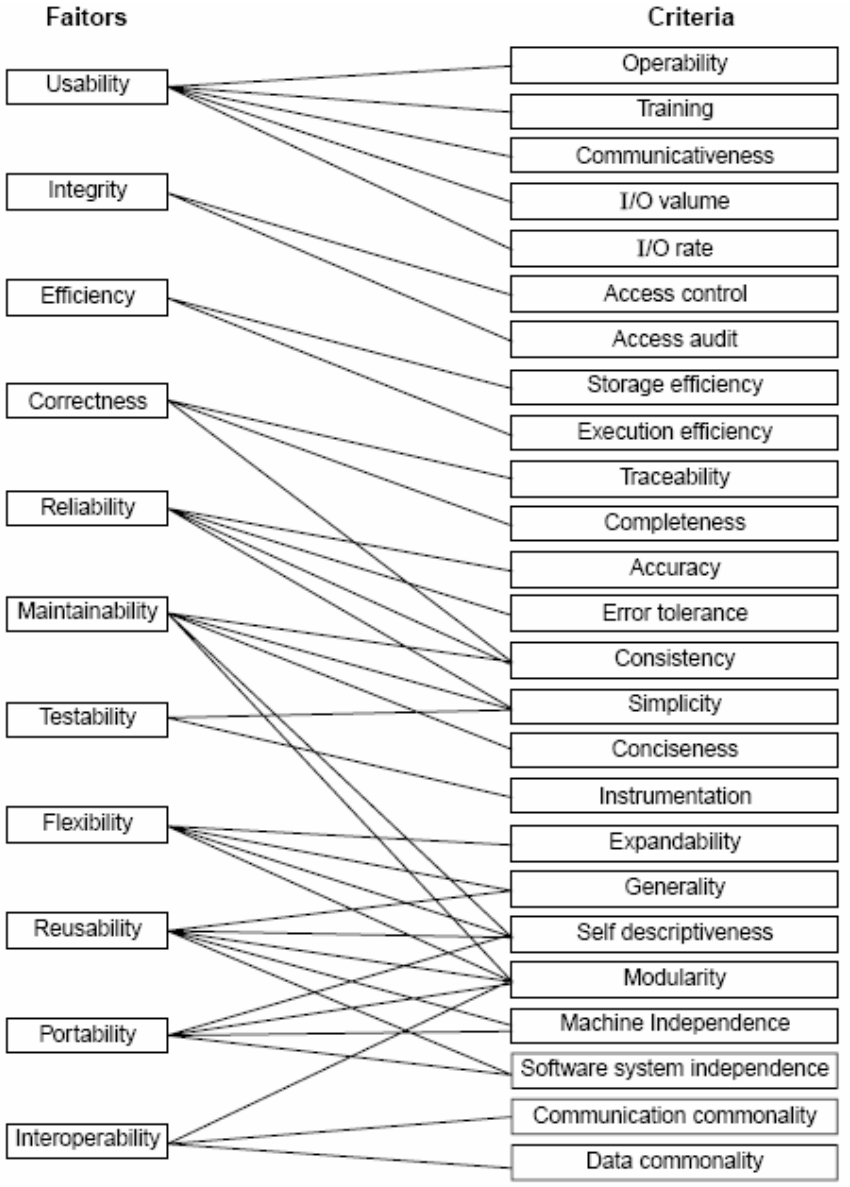


Fig 7.10: McCall's quality model

Software Reliability

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

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

- Boehm Software Quality Model

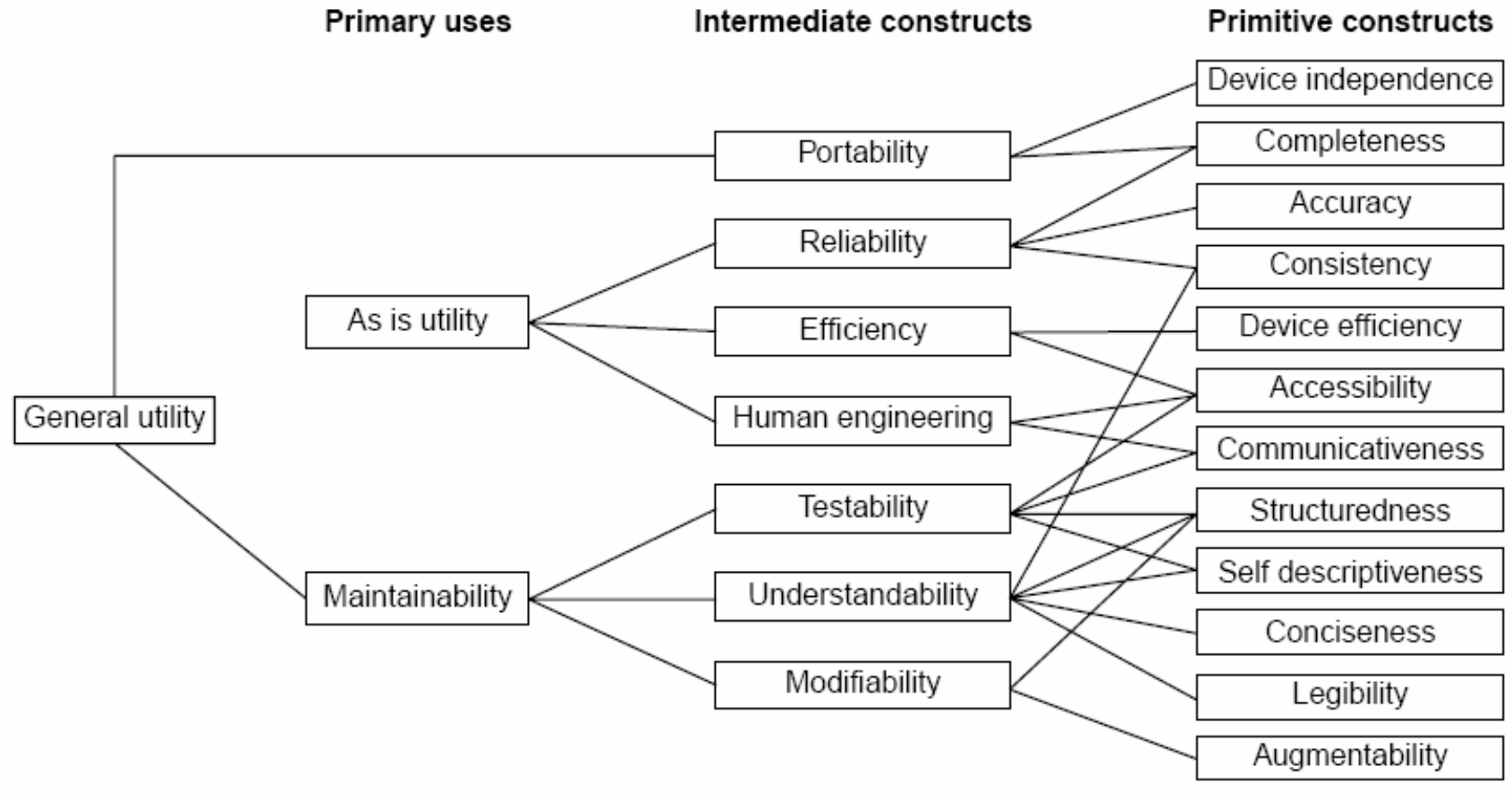


Fig.7.11: The Boehm software quality model

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

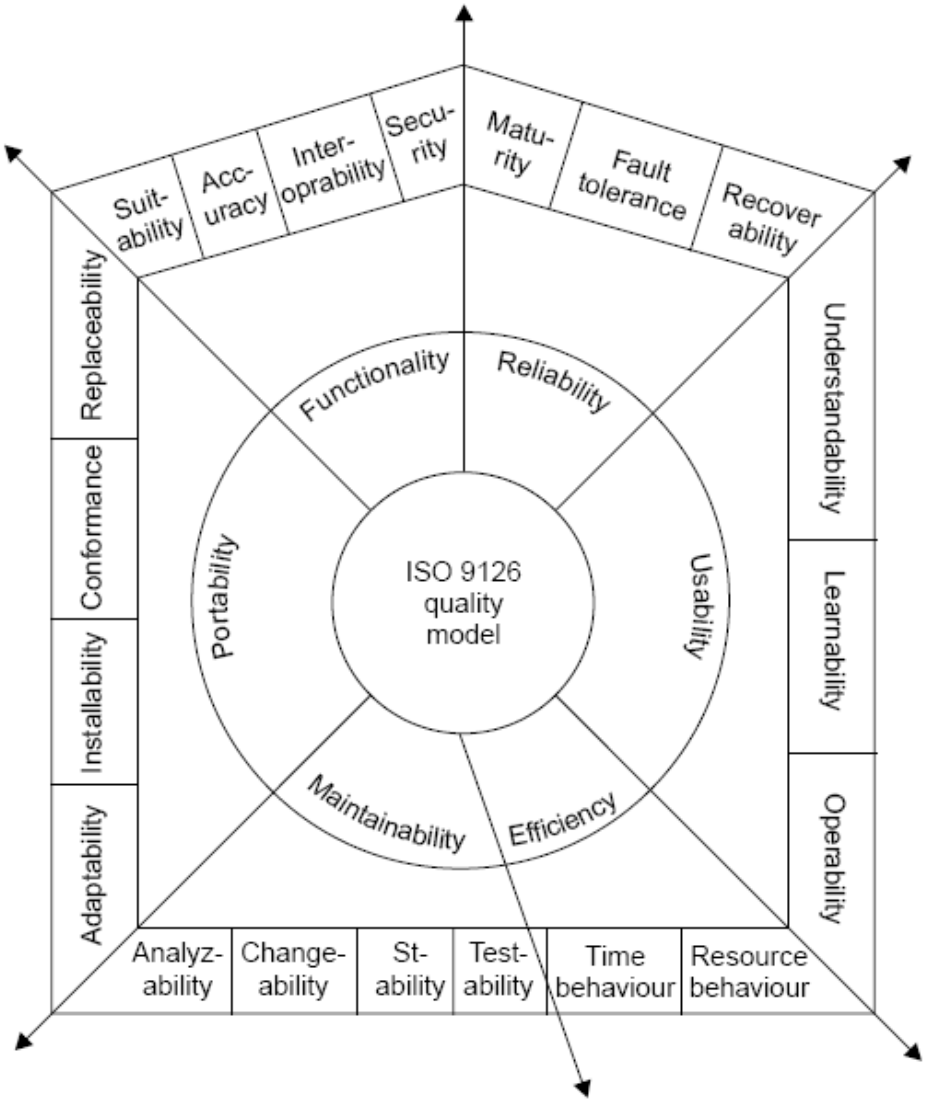
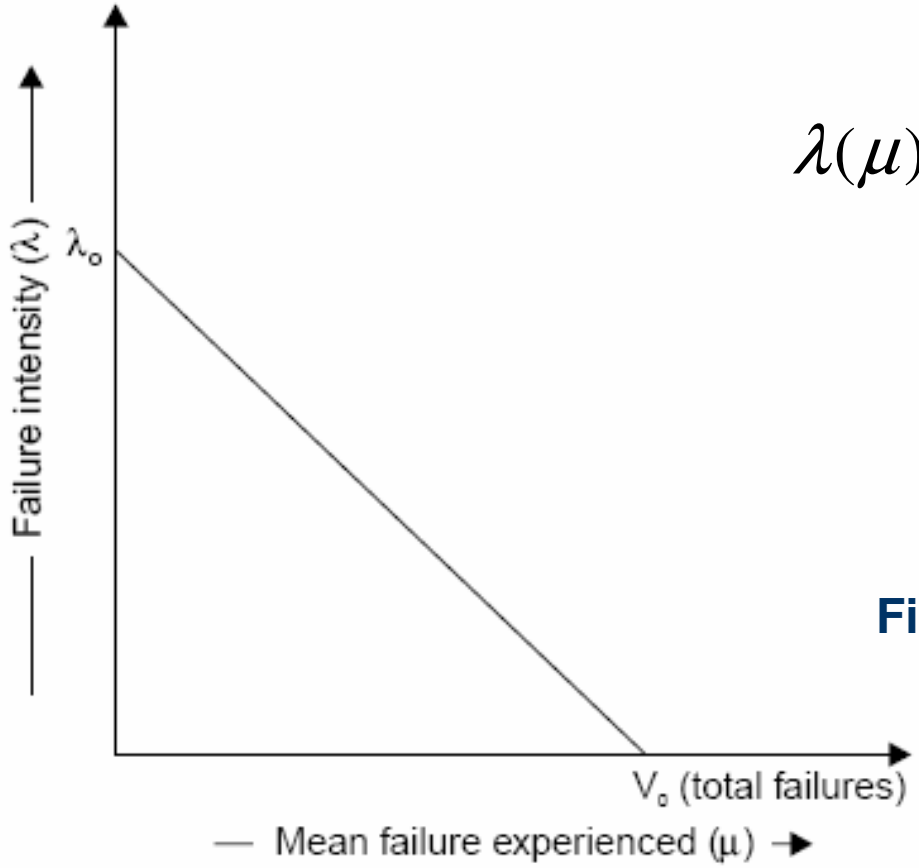


Fig.7.12: ISO 9126 quality model

Software Reliability Models

- Basic Execution Time Model



$$\lambda(\mu) = \lambda_0\left(1 - \frac{\mu}{V_0}\right) \qquad (1)$$

Fig.7.13: Failure intensity λ as a function of μ for basic model

Software Reliability

$$\frac{d\lambda}{d\mu} = \frac{-\lambda_0}{V_0} \qquad (2)$$

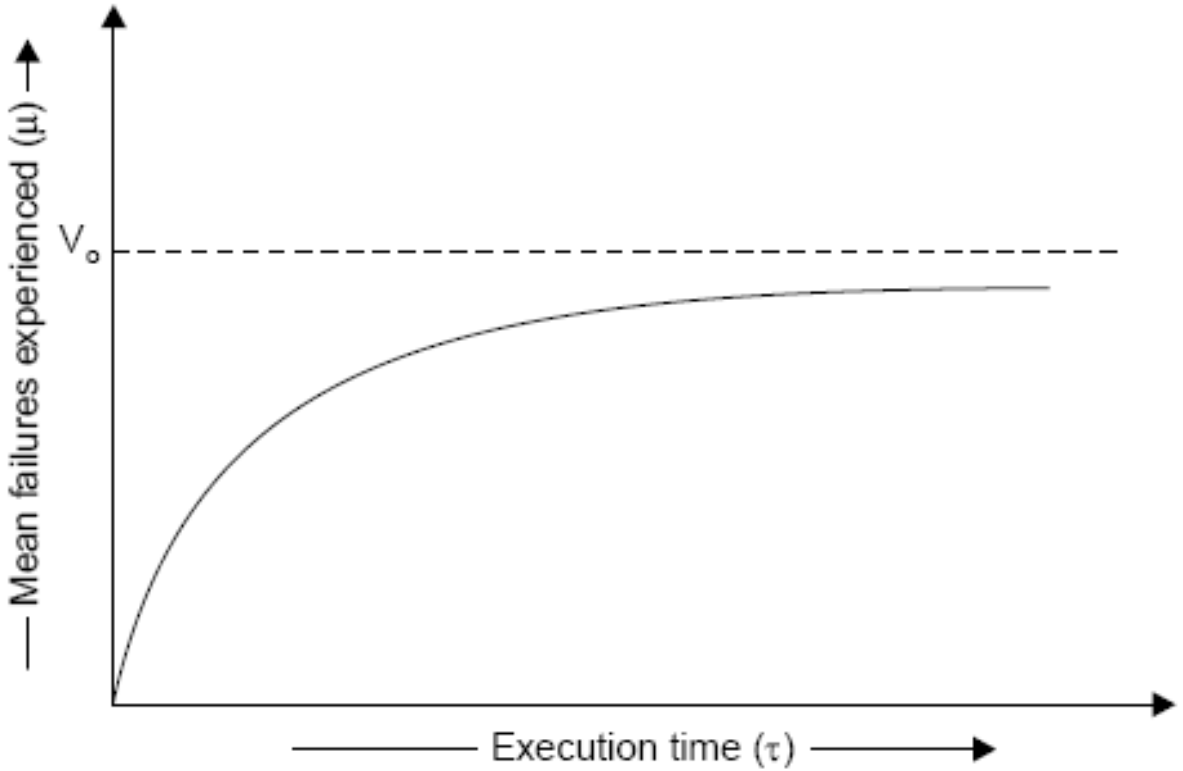


Fig.7.14: Relationship between τ & μ for basic model

Software Reliability

For a derivation of this relationship, equation 1 can be written as:

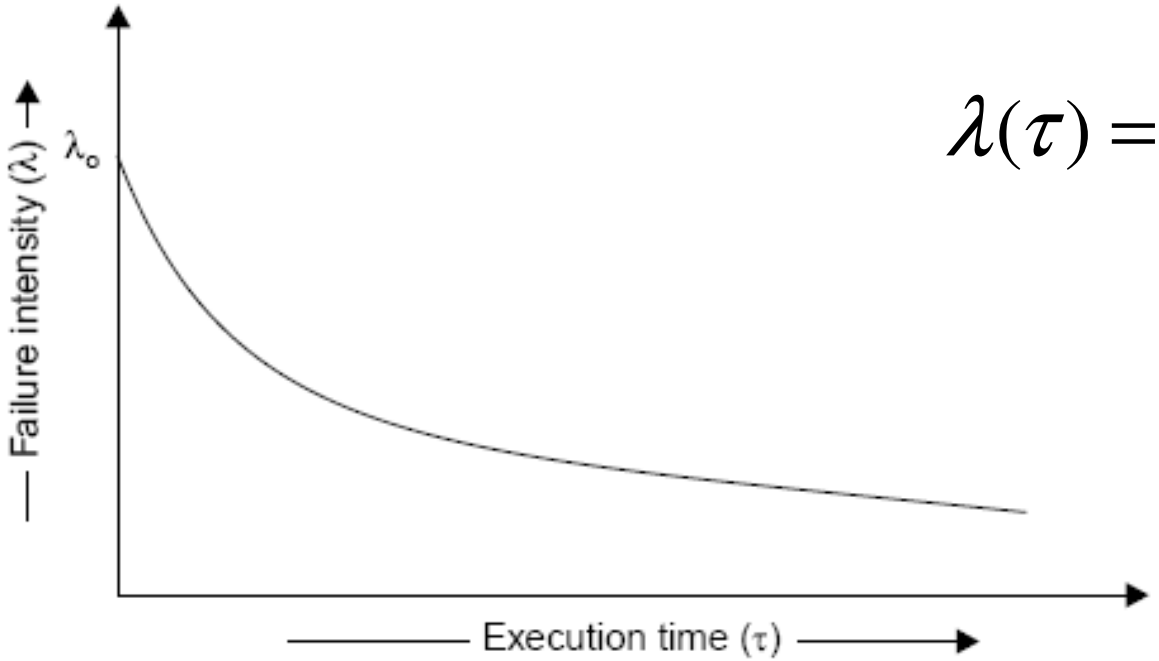
$$\frac{d\mu(\tau)}{d\tau} = \lambda_0\left(1 - \frac{\mu(\tau)}{V_0}\right)$$

The above equation can be solved for $\mu(\tau)$ and result in :

$$\mu(\tau) = V_0\left(1 - \exp\left(\frac{-\lambda_0\tau}{V_0}\right)\right) \quad \textbf{(3)}$$

Software Reliability

The failure intensity as a function of execution time is shown in figure given below



$$\lambda(\tau) = \lambda_0 \exp\left(\frac{-\lambda_0 \tau}{V_0}\right)$$

Fig.7.15: Failure intensity versus execution time for basic model

Software Reliability

- Derived quantities

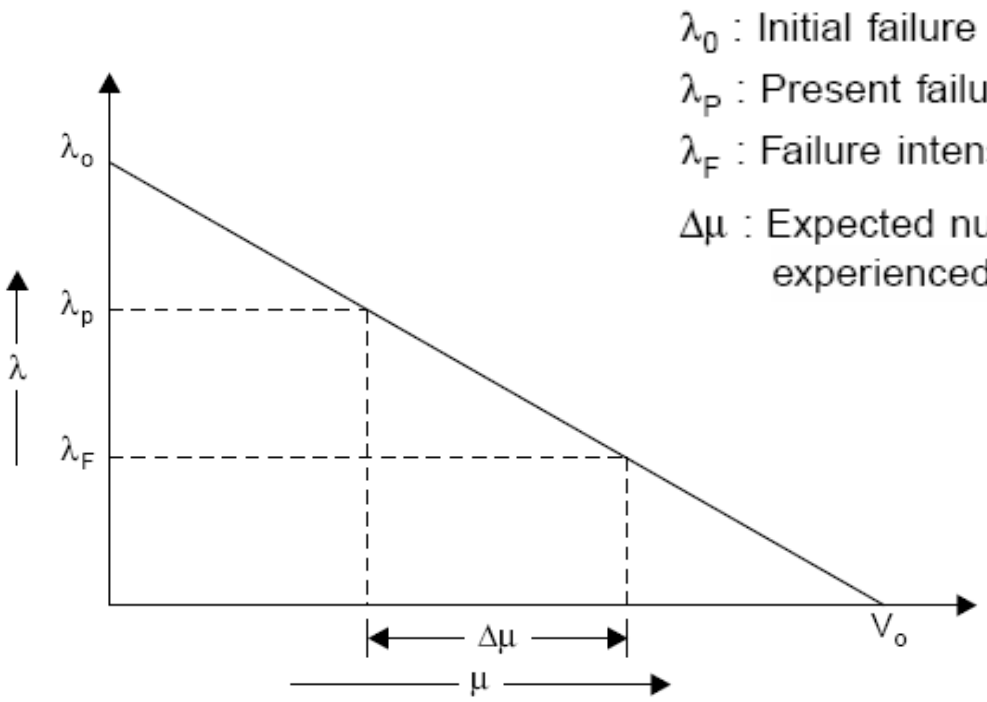


Fig.7.16: Additional failures required to be experienced to reach the objective

Software Reliability

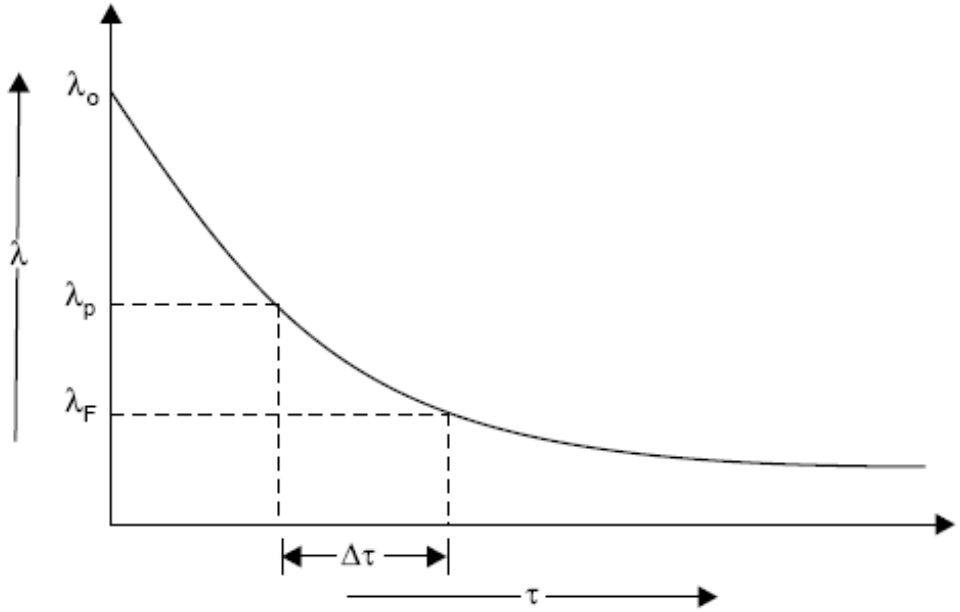


Fig.7.17: Additional time required to reach the objective

This can be derived in mathematical form as:

$$\Delta\tau = \frac{V_0}{\lambda_0} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$$

Software Reliability

Example- 7.1

Assume that a program will experience 200 failures in infinite time. It has now experienced 100. The initial failure intensity was 20 failures/CPU hr.

(i) Determine the current failure intensity.

(ii) Find the decrement of failure intensity per failure.

(iii) Calculate the failures experienced and failure intensity after 20 and 100 CPU hrs. of execution.

(iv) Compute addition failures and additional execution time required to reach the failure intensity objective of 5 failures/CPU hr.

Use the basic execution time model for the above mentioned calculations.

Solution

Here V_o=200 failures

$$\mu = 100 \text{ failures}$$

$$\lambda_0 = 20 \text{ failures/CPU hr.}$$

(i) Current failure intensity:

$$\lambda(\mu) = \lambda_0\left(1 - \frac{\mu}{V_0}\right)$$

$$= 20\left(1 - \frac{100}{200}\right) = 20(1 - 0.5) = 10 \text{ failures/CPU hr}$$

(ii) Decrement of failure intensity per failure can be calculated as:

$$\frac{d\lambda}{d\mu} = \frac{-\lambda_0}{V_0} = -\frac{20}{200} = -0.1/\text{CPU hr.}$$

(iii) (a) Failures experienced & failure intensity after 20 CPU hr:

$$\mu(\tau) = V_0\left(1 - \exp\left(\frac{-\lambda_0\tau}{V_0}\right)\right)$$

$$= 200\left(1 - \exp\left(\frac{-20\times 20}{200}\right)\right) = 200(1 - \exp(1-2))$$

$$= 200(1 - 0.1353) \approx 173 \text{failures}$$

Software Reliability

$$\lambda(\tau) = \lambda_0 \exp\left(\frac{-\lambda_0 \tau}{V_0}\right)$$

$$= 20 \exp\left(\frac{-20 \times 20}{200}\right) = 20 \exp(-2) = 2.71 \, failures / CPU \ \ hr$$

(b) Failures experienced & failure intensity after 100 CPU hr:

$$\mu(\tau) = V_0\left(1 - \exp\left(\frac{-\lambda_0 \tau}{V_0}\right)\right)$$

$$= 200\left(1 - \exp\left(\frac{-20 \times 100}{200}\right)\right) = 200 \text{ failures(almost)}$$

$$\lambda(\tau) = \lambda_0 \exp\left(\frac{-\lambda_0 \tau}{V_0}\right)$$

$$= 20 \exp\left(\frac{-20 \times 100}{200}\right) = 0.000908 \ failures / CPU \ hr$$

(iv) Additional failures $(\Delta\mu)$ required to reach the failure intensity objective of 5 failures/CPU hr.

$$\Delta\mu = \left(\frac{V_0}{\lambda_0}\right)(\lambda_P - \lambda_F) = \left(\frac{200}{20}\right)(10-5) = 50 \text{ failures}$$

Software Reliability

Additional execution time required to reach failure intensity objective of 5 failures/CPU hr.

$$\Delta\tau = \left(\frac{V_0}{\lambda_0}\right) Ln\left(\frac{\lambda_P}{\lambda_F}\right)$$

$$= \frac{200}{20} Ln\left(\frac{10}{5}\right) = 6.93 \text{ CPU hr.}$$

Software Reliability

- Logarithmic Poisson Execution Time Model

  Failure Intensity

$$\lambda(\mu) = \lambda_0 \exp(-\theta\mu)$$

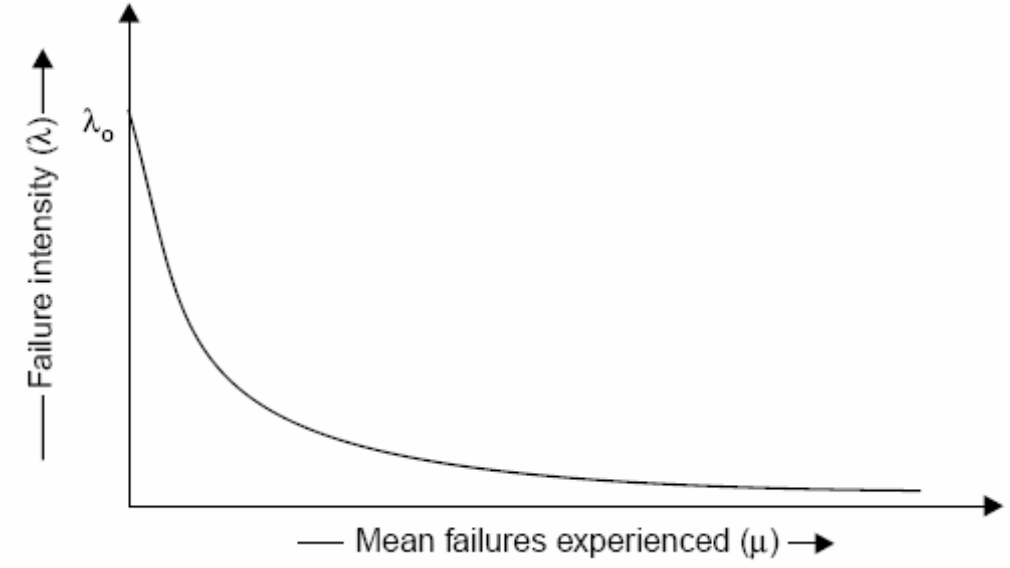


Fig.7.18: Relationship between μ & λ

Software Reliability

$$\frac{d\lambda}{d\mu} = -\lambda_0 \theta \exp(-\mu\theta)$$

$$\frac{d\lambda}{d\mu} = -\theta\lambda$$

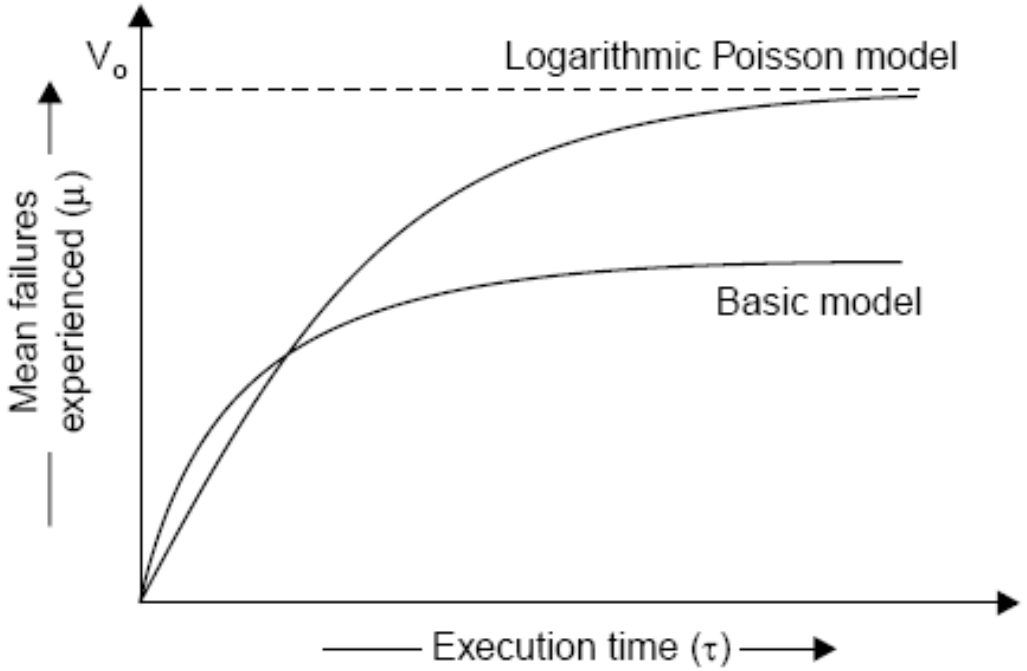


Fig.7.19: Relationship between

Software Reliability

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

$$\lambda(\tau) = \lambda_0 / (\lambda_0 \theta \tau + 1)$$

$$\Delta\mu = \frac{1}{\theta} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$$

$$\Delta\tau = \frac{1}{\theta}\left[\frac{1}{\lambda_F} - \frac{1}{\lambda_P}\right] \quad (4)$$

λ_P = Present failure intensity

λ_F = Failure intensity objective

Software Reliability

Example- 7.2

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

(i) Determine the current failure intensity.

(ii) Calculate the decrement of failure intensity per failure.

(iii) Find the failures experienced and failure intensity after 20 and 100 CPU hrs. of execution.

(iv) Compute the additional failures and additional execution time required to reach the failure intensity objective of 2 failures/CPU hr.

Use Logarithmic Poisson execution time model for the above mentioned calculations.

Software Reliability

Solution

$$\lambda_0 = 20 \text{ failures/CPU hr.}$$

$$\mu = 100 \text{ failures}$$

$$\theta = 0.02 / \text{failures}$$

(i) Current failure intensity:

$$\lambda(\mu) = \lambda_0 \exp(-\theta\mu)$$

= 20 exp (-0.02 x 100)

= 2.7 failures/CPU hr.

Software Reliability

(ii) Decrement of failure intensity per failure can be calculated as:

$$\frac{d\lambda}{d\mu} = -\theta\lambda$$

= -.02 x 2.7 = -.054/CPU hr.

(iii) (a) Failures experienced & failure intensity after 20 CPU hr:

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

$$= \frac{1}{0.02} Ln(20 \times 0.02 \times 20 + 1) = 109 \ failures$$

Software Reliability

$$\lambda(\tau) = \lambda_0 / (\lambda_0 \theta \tau + 1)$$

$$= (20) / (20 \times .02 \times 20 + 1) = 2.22 \ failures / CPU \ hr.$$

(b) Failures experienced & failure intensity after 100 CPU hr:

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

$$= \frac{1}{0.02} Ln(20 \times 0.02 \times 100 + 1) = 186 \ failures$$

$$\lambda(\tau) = \lambda_0 / (\lambda_0 \theta \tau + 1)$$

$$= (20) / (20 \times .02 \times 100 + 1) = 0.4878 \ failures / CPU \ hr.$$

Software Reliability

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

$$\Delta\mu = \frac{1}{\theta} Ln \frac{\lambda_P}{\lambda_F} = \frac{1}{0.02} Ln\left(\frac{2.7}{2}\right) = 15 \text{ failures}$$

$$\Delta\tau = \frac{1}{\theta}\left[\frac{1}{\lambda_F} - \frac{1}{\lambda_P}\right] = \frac{1}{0.02}\left[\frac{1}{2} - \frac{1}{2.7}\right] = 6.5 \text{ CPU hr.}$$

Example- 7.3

The following parameters for basic and logarithmic Poisson models are given:

(a) Determine the addition failures and additional execution time required to reach the failure intensity objective of 5 failures/CPU hr. for both models.

(b) Repeat this for an objective function of 0.5 failure/CPU hr. Assume that we start with the initial failure intensity only.

Basic execution time model	Logarithmic Poisson execution time model
$\lambda_o = 10$ failures/CPU hr	$\lambda_o = 30$ failures/CPU hr
$V_o = 100$ failures	$\theta = 0.25$/failure

Solution

(a) (i) Basic execution time model

$$\Delta\mu = \frac{V_0}{\lambda_0}(\lambda_P - \lambda_F)$$

$$= \frac{100}{10}(10-5) = 50 \text{ failures}$$

λ_P (Present failure intensity) in this case is same as λ_0 (initial failure intensity).

Now,

$$\Delta\tau = \frac{V_0}{\lambda_0} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$$

$$= \frac{100}{10} Ln\left(\frac{10}{5}\right) = 6.93 \text{ CPU hr.}$$

(ii) Logarithmic execution time model

$$\Delta\mu = \frac{1}{\theta} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$$

$$= \frac{1}{0.025} Ln\left(\frac{30}{5}\right) = 71.67 \text{ Failures}$$

$$\Delta\tau = \frac{1}{\theta}\left(\frac{1}{\lambda_F} - \frac{1}{\lambda_P}\right)$$

$$= \frac{1}{0.025} Ln\left(\frac{1}{5} - \frac{1}{30}\right) = 6.66 \text{ CPU hr.}$$

Software Reliability

Logarithmic model has calculated more failures in almost some duration of execution time initially.

(b) Failure intensity objective (λ_F) = 0.5 failures/CPU hr.

(i) Basic execution time model

$$\Delta\mu = \frac{V_0}{\lambda_0}(\lambda_P - \lambda_F)$$

$$= \frac{100}{10}(10 - 0.5) = 95 \ failures$$

$$\Delta\tau = \frac{V_0}{\lambda_0} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$$

$$= \frac{100}{10} Ln\left(\frac{10}{0.05}\right) = 30 \ CPU/hr$$

Software Reliability

(ii) Logarithmic execution time model

$$\Delta\mu = \frac{1}{\theta} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$$

$$= \frac{1}{0.025} Ln\left(\frac{30}{0.5}\right) = 164 \ failures$$

$$\Delta\tau = \frac{1}{\theta}\left(\frac{1}{\lambda_F} - \frac{1}{\lambda_P}\right)$$

$$= \frac{1}{0.025}\left(\frac{1}{0.5} - \frac{1}{30}\right) = 78.66 \ CPU/hr$$

Software Reliability

- Calendar Time Component

The calendar time component is based on a debugging process model. This model takes into account:

1. resources used in operating the program for a given execution time and processing an associated quantity of failure.
2. resources quantities available, and
3. the degree to which a resource can be utilized (due to bottlenecks) during the period in which it is limiting.

Table 7.7 will help in visualizing these different aspects of the resources, and the parameters that result.

Resource usage

	Usage parameters requirements per		Planned parameters	
Resource	**CPU hr**	**Failure**	**Quantities available**	**Utilisation**
Failure identification personnel	θ_I	μ_I	P_I	1
Failure correction personnel	0	μ_f	P_f	P_f
Computer time	θc	μ_c	P_c	P_c

Fig. : Calendar time component resources and parameters

Software Reliability

Hence, to be more precise, we have

$$X_C = \mu_c \Delta\mu + \theta_c \Delta\tau$$ (for computer time)

$$X_f = \mu_f \Delta\mu$$ (for failure correction)

$$X_I = \mu_I \Delta\mu + \theta_I \Delta\tau$$ (for failure identification)

$$dx_T / d\tau = \theta_r + \mu_r \lambda$$

Software Reliability

Calendar time to execution time relationship

$$dt / d\tau = (1 / P_r p_r) dx_T / d\tau$$

$$dt / d\tau = (\theta_r + \mu_r \lambda) / P_r p_r$$

Software Reliability

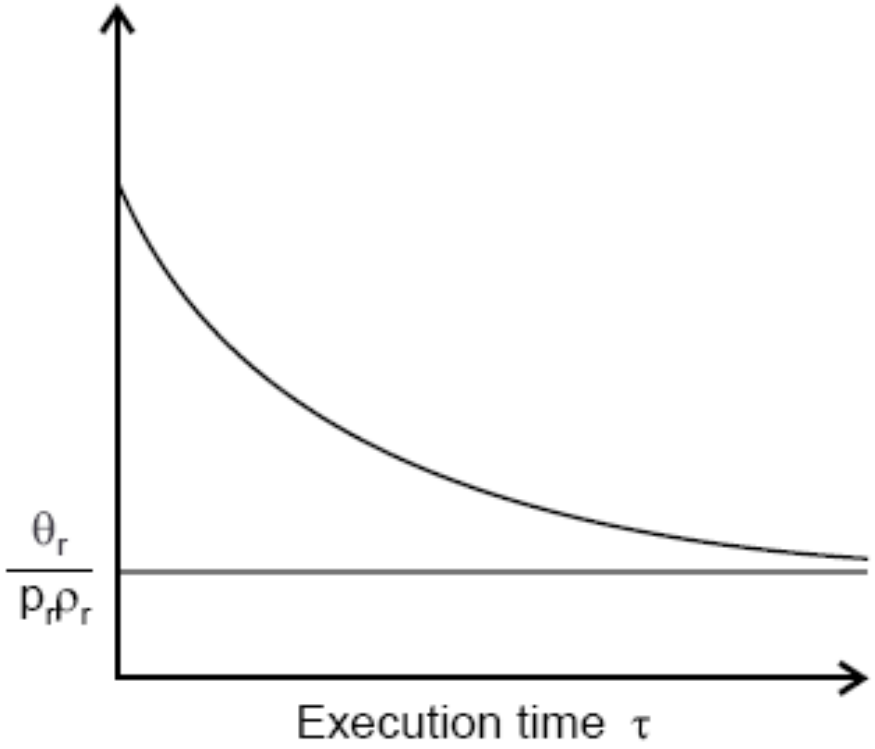


Fig.7.20: Instantaneous calendar time to execution time ratio

Software Reliability

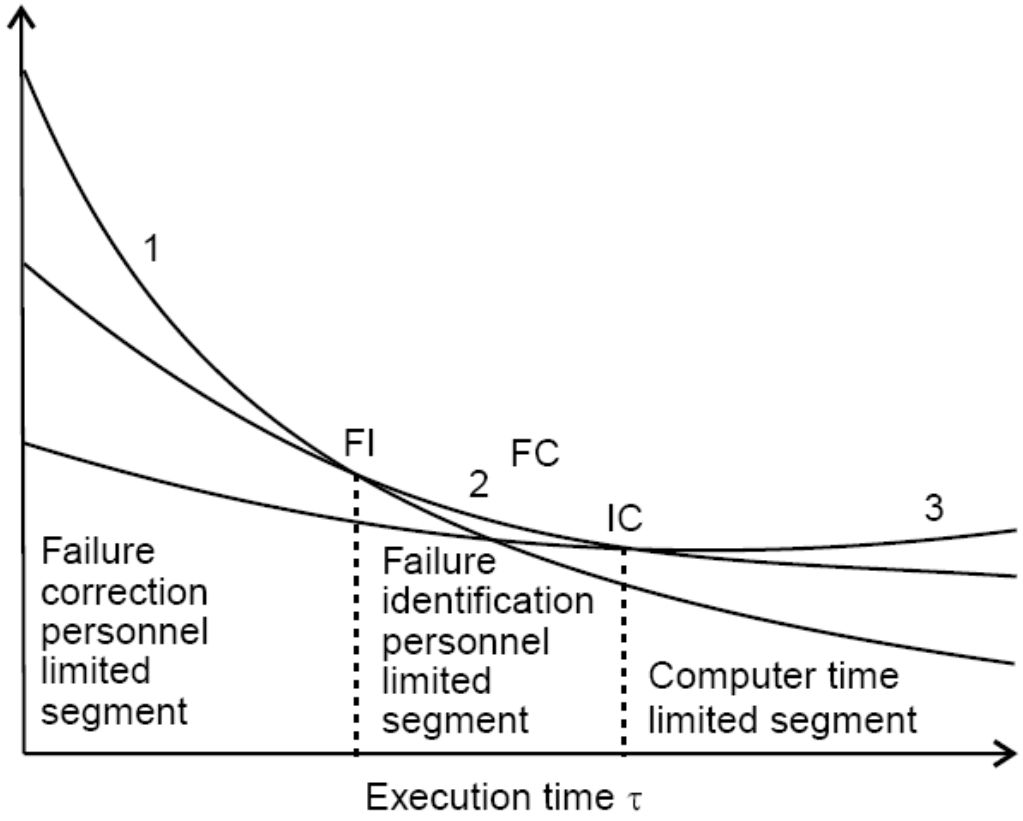


Fig.7.21: Calendar time to execution time ratio for different limiting resources

Software Reliability

Example- 7.4

A team run test cases for 10 CPU hrs and identifies 25 failures. The effort required per hour of execution time is 5 person hr. Each failure requires 2 hr. on an average to verify and determine its nature. Calculate the failure identification effort required.

Solution

As we know, resource usage is:

$$X_r = \theta_r \tau + \mu_r \mu$$

Here $\theta_r = 15$ person hr. $\mu = 25$ failures

$\tau = 10$ CPU hrs. $\mu_r = 2$ hrs./failure

Hence, $X_r = 5\ (10) + 2\ (25)$

$= 50 + 50 = 100$ person hr.

Example- 7.5

Initial failure intensity (λ_0) for a given software is 20 failures/CPU hr. The failure intensity objective (λ_F) of 1 failure/CPU hr. is to be achieved. Assume the following resource usage parameters.

Resource Usage	Per hour	Per failure
Failure identification effort	2 Person hr.	1 Person hr.
Failure Correction effort	0	5 Person hr.
Computer time	1.5 CPU hr.	1 CPU hr.

Software Reliability

(a) What resources must be expended to achieve the reliability improvement? Use the logarithmic Poisson execution time model with a failure intensity decay parameter of 0.025/failure.

(b) If the failure intensity objective is cut to half, what is the effect on requirement of resources ?

Solution

(a) $$\Delta\mu = \frac{1}{\theta} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$$

$$= \frac{1}{0.025} Ln\left(\frac{20}{1}\right) = 119 \text{ failures}$$

$$\Delta\tau = \frac{1}{\theta}\left(\frac{1}{\lambda_F} - \frac{1}{\lambda_P}\right)$$

$$= \frac{1}{0.025}\left(\frac{1}{1} - \frac{1}{20}\right) = \frac{1}{0.025}(1 - 0.05) = 38 \text{ CPU hrs.}$$

Software Reliability

Hence $$X_1 = \mu_1 \Delta\mu + \theta_1 \Delta\tau$$

= 1 (119) + 2 (38) = 195 Person hrs.

$$X_F = \mu_F \Delta\mu$$

= 5 (119) = 595 Person hrs.

$$X_C = \mu_c \Delta\mu + \theta_c \Delta\tau$$

= 1 (119) + (1.5) (38) = 176 CPU hr.

Software Reliability

(b) $\lambda_F = 0.5$ failures/CPU hr.

$$\Delta\mu = \frac{1}{0.025} Ln\left(\frac{20}{0.5}\right) = 148 \text{ failures}$$

$$\Delta\tau = \frac{1}{0.025}\left(\frac{1}{0.5} - \frac{1}{20}\right) = 78 \text{ CPU hr.}$$

So, $X_I = 1\ (148) + 2\ (78) = 304$ Person hrs.

$X_F = 5\ (148) = 740$ Person hrs.

$X_C = 1\ (148) + (1.5)(78) = 265$ CPU hrs.

Software Reliability

Hence, if we cut failure intensity objective to half, resources requirements are not doubled but they are some what less. Note that $\Delta\tau$ is approximately doubled but increases logarithmically. Thus, the resources increase will be between a logarithmic increase and a linear increase for changes in failure intensity objective.

Software Reliability

Example- 7.6

A program is expected to have 500 faults. It is also assumed that one fault may lead to one failure only. The initial failure intensity was 2 failures/CPU hr. The program was to be released with a failure intensity objective of 5 failures/100 CPU hr. Calculated the number of failure experienced before release.

Software Reliability

Solution

The number of failure experienced during testing can be calculated using the equation mentioned below:

$$\Delta\mu = \frac{V_0}{\lambda_0}(\lambda_P - \lambda_F)$$

Here $V_0 = 500$ because one fault leads to one failure

$\lambda_0 = 2$ failures/CPU hr.

$\lambda_F = 5$ failures/100 CPU hr.

$= 0.05$ failures/CPU hr.

Software Reliability

So $$\Delta\mu = \frac{500}{2}(2 - 0.05)$$

= 487 failures

Hence 13 faults are expected to remain at the release instant of the software.

Software Reliability

- The Jelinski-Moranda Model

$$\lambda(t) = \phi(N - i + 1)$$

where

ϕ = Constant of proportionality

N = Total number of errors present

I = number of errors found by time interval t_i

Software Reliability

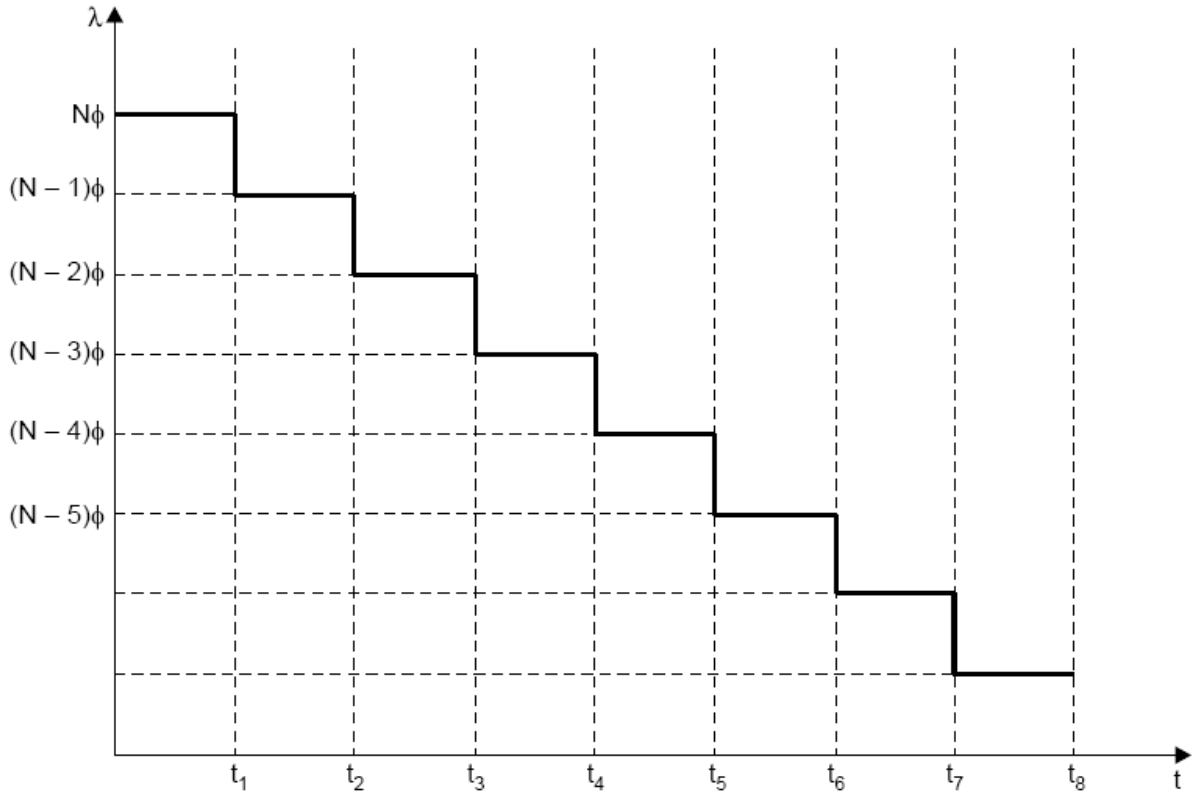


Fig.7.22: Relation between t & λ

Software Reliability

Example- 7.7

There are 100 errors estimated to be present in a program. We have experienced 60 errors. Use Jelinski-Moranda model to calculate failure intensity with a given value of ϕ=0.03. What will be failure intensity after the experience of 80 errors?

Software Reliability

Solution

N = 100 errors

i = 60 failures

ϕ = 0.03

We know $\lambda(t) = 0.03(100 - 60 + 1)$

= 0.03(100-60+1)

= 1.23 failures/CPU hr.

After 80 failures $\lambda(t) = 0.03(100 - 80 + 1)$

= 0.63 failures/CPU hr.

Hence, there is continuous decrease in the failure intensity as the number of failure experienced increases.

Software Reliability

- The Bug Seeding Model

The bug seeding model is an outgrowth of a technique used to estimate the number of animals in a wild life population or fish in a pond.

$$\frac{N_t}{N + N_t} = \frac{n_t}{n + n_t}$$

$$\hat{N} = \frac{n}{n_t} N_t$$

$$N = \frac{n}{n_s} N_s$$

Software Reliability

- Capability Maturity Model

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

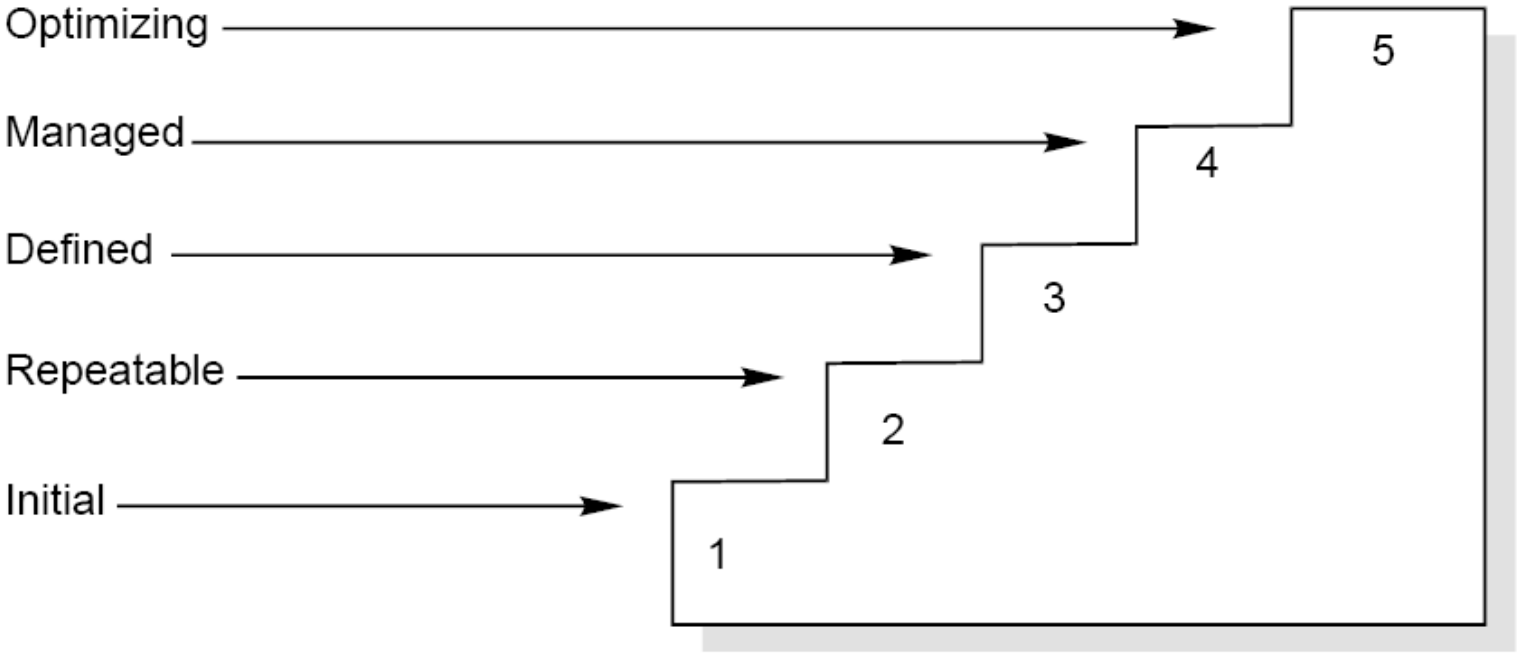


Fig.7.23: Maturity levels of CMM

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

Maturity 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

- Key Process Areas

The key process areas at level 2 focus on the software project's concerns related to establishing basic project management controls, as summarized below:

Requirements Management (RM)	Establish a common relationship between the customer requirements and the developers in order to understand the requirements of the project.
Software Project Planning (PP)	Establish reasonable plans for performing the software engineering and for managing the software project.
Software Project Tracking and Oversight (PT)	Establish adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.
Software Subcontract Management (SM)	Select qualified software subcontractors and manage them effectively.
Software Quality Assurance (QA)	Provide management with appropriate visibility into the process being used by the software project and of the products being built.
Software Configuration Management (CM)	Establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

Software Reliability

The key process areas at level 3 address both project and organizational issues, as summarized below:

Organization Process Focus (PF)	Establish the organizational responsibility for software process activities that improve the organization's overall software process capability.
Organization Process Definition (PD)	Develop and maintain a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organization.
Training Program (TP)	Develop the skills and knowledge of individuals so that they can perform their roles effectively and efficiently.
Integrated Software Management (IM)	Integrate the software engineering and management activities into a coherent, defined software process that is tailored from the organization's standard software process and related process assets.

(Contd.)...

Software Reliability

Software Product Engineering (PE)	Consistently perform a well-defined engineering process that integrates all the software engineering activities to produce correct, consistent software products effectively and efficiently.
Inter group Coordination (IC)	Establish a means for the software engineering group to participate actively with the other engineering groups so the project is better able to satisfy the customer's needs effectively and efficiently.
Peer Reviews (PR)	Remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of the defects that can be prevented.

Software Reliability

The key process areas at level 4 focus on establishing a quantitative understanding of both the software process and the software work products being built, as summarized below:

Quantitative Process Management (QP)	Control the process performance of the software project quantitatively.
Software Quality Management (QM)	Develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals.

Software Reliability

The key process areas at level 5 cover the issues that both the organization and the projects must address to implement continuous and measurable software process improvement, as summarized below:

Defect Prevention (DP)	Identify the causes of defects and prevent them from recurring.
Technology Change Management (TM)	Identify beneficial new technologies (i.e., tools, methods, and processes) and transfer them into the organization in an orderly manner.
Process Change Management (PC)	Continually improve the software processes used in the organization with the intent of improving software quality, increasing productivity, and decreasing the cycle time for product development.

Software Reliability

- Common Features

Commitment to Perform (CO)	Describes the actions the organizations must take to ensure that the process is established and will endure. It includes practices on policy and leadership.
Ability to Perform (AB)	Describes the preconditions that must exist in the project or organization to implement the software process competently. It includes practices on resources, organizational structure, training, and tools.
Activities Performed (AC)	Describes the role and procedures necessary to implement a key process area. It includes practices on plans, procedures, work performed, tracking, and corrective action.
Measurement and Analysis (ME)	Describes the need to measure the process and analyze the measurements. It includes examples of measurements.
Verifying Implementation (VE)	Describes the steps to ensure that the activities are performed in compliance with the process that has been established. It includes practices on management reviews and audits.

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 document 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"**.

Multiple Choice Questions

Note: Choose most appropriate answer of the following questions:

7.1 Which one is not a phase of "bath tub curve" of hardware reliability

(a) Burn-in
(b) Useful life
(c) Wear-out
(d) Test-out

7.2 Software reliability is

(a) the probability of failure free operation of a program for a specified time in a specified environment

(b) the probability of failure of a program for a specified time in a specified environment

(c) the probability of success of a program for a specified time in any environment

(d) None of the above

7.3 Fault is

(a) Defect in the program
(b) Mistake in the program
(c) Error in the program
(d) All of the above

7.4 One fault may lead to

(a) one failure
(b) two failures
(c) many failures
(d) all of the above

Multiple Choice Questions

7.5 Which 'time' unit is not used in reliability studies

(a) Execution time (b) Machine time

(c) Clock time (d) Calendar time

7.6 Failure occurrences can be represented as

(a) time to failure (b) time interval between failures

(c) failures experienced in a time interval (d) All of the above

7.7 Maximum possible value of reliability is

(a) 100 (b) 10

(c) 1 (d) 0

7.8 Minimum possible value of reliability is

(a) 100 (b) 10

(c) 1 (d) 0

7.9 As the reliability increases, failure intensity

(a) decreases (b) increases

(c) no effect (d) None of the above

Multiple Choice Questions

7.10 If failure intensity is 0.005 failures/hour during 10 hours of operation of a software, its reliability can be expressed as

(a) 0.10 (b) 0.92

(c) 0.95 (d) 0.98

7.11 Software Quality is

(a) Conformance to requirements (b) Fitness for the purpose

(c) Level of satisfaction (d) All of the above

7.12 Defect rate is

(a) number of defects per million lines of source code
(b) number of defects per function point
(c) number of defects per unit of size of software
(d) All of the above

7.13 How many product quality factors have been proposed in McCall quality model?

(a) 2 (b) 3

(c) 11 (d) 6

Multiple Choice Questions

7.14 Which one is not a product quality factor of McCall quality model?

(a) Product revision
(b) Product operation
(c) Product specification
(d) Product transition

7.15 The second level of quality attributes in McCall quality model are termed as

(a) quality criteria
(b) quality factors
(c) quality guidelines
(d) quality specifications

7.16 Which one is not a level in Boehm software quality model ?

(a) Primary uses
(b) Intermediate constructs
(c) Primitive constructs
(d) Final constructs

7.17 Which one is not a software quality model?

(a) McCall model
(b) Boehm model
(c) ISO 9000
(d) ISO 9126

7.18 Basic execution time model was developed by

(a) Bev.Littlewood
(b) J.D.Musa
(c) R.Pressman
(d) Victor Baisili

Multiple Choice Questions

7.19 NHPP stands for

(a) Non Homogeneous Poisson Process (b) Non Hetrogeneous Poisson Process

(c) Non Homogeneous Poisson Product (d) Non Hetrogeneous Poisson Product

7.20 In Basic execution time model, failure intensity is given by

$(a)\ \lambda(\mu) = \lambda_0\left(1 - \frac{\mu^2}{V_0}\right)$ $(b)\ \lambda(\mu) = \lambda_0\left(1 - \frac{\mu}{V_0}\right)$

$(c)\ \lambda(\mu) = \lambda_0\left(1 - \frac{V_0}{\mu}\right)$ $(d)\ \lambda(\mu) = \lambda_0\left(1 - \frac{V_0}{\mu^2}\right)$

7.21 In Basic execution time model, additional number of failures required to achieve a failure intensity objective $(\Delta\mu)$ is expressed as

$(a)\ \Delta\mu = \frac{V_0}{\lambda_0}(\lambda_P - \lambda_F)$ $(b)\ \Delta\mu = \frac{V_0}{\lambda_0}(\lambda_F - \lambda_P)$

$(c)\ \Delta\mu = \frac{\lambda_0}{V_0}(\lambda_F - \lambda_P)$ $(d)\ \Delta\mu = \frac{\lambda_0}{V_0}(\lambda_P - \lambda_F)$

Multiple Choice Questions

7.22 In Basic execution time model, additional time required to achieve a failure intensity objective $(\Delta\tau)$ is given as

$(a)\ \Delta\tau = \frac{\lambda_0}{V_0} Ln\left(\frac{\lambda_F}{\lambda_P}\right)$ $(b)\ \Delta\tau = \frac{\lambda_0}{V_0} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$

$(c)\ \Delta\tau = \frac{V_0}{\lambda_0} Ln\left(\frac{\lambda_F}{\lambda_P}\right)$ $(d)\ \Delta\tau = \frac{V_0}{\lambda_0} Ln\left(\frac{\lambda_P}{\lambda_F}\right)$

7.23 Failure intensity function of Logarithmic Poisson execution model is given as

$(a)\ \lambda(\mu) = \lambda_0 LN(-\theta\mu)$ $(b)\ \lambda(\mu) = \lambda_0 \exp(\theta\mu)$

$(c)\ \lambda(\mu) = \lambda_0 \exp(-\theta\mu)$ $(d)\ \lambda(\mu) = \lambda_0 \log(-\theta\mu)$

7.24 In Logarithmic Poisson execution model, 'θ' is known as

(a) Failure intensity function parameter (b) Failure intensity decay parameter

(c) Failure intensity measurement (d) Failure intensity increment parameter

Multiple Choice Questions

7.25 In jelinski-Moranda model, failure intensity is defined aseneous Poisson Product

$(a)\ \lambda(t) = \phi(N - i + 1)$ $(b)\ \lambda(t) = \phi(N + i + 1)$

$(c)\ \lambda(t) = \phi(N + i - 1)$ $(d)\ \lambda(t) = \phi(N - i - 1)$

7.26 CMM level 1 has

(a) 6 KPAs (b) 2 KPAs

(c) 0 KPAs (d) None of the above

7.27 MTBF stands for

(a) Mean time between failure (b) Maximum time between failures

(c) Minimum time between failures (d) Many time between failures

7.28 CMM model is a technique to

(a) Improve the software process (b) Automatically develop the software

(c) Test the software (d) All of the above

7.29 Total number of maturing levels in CMM are

(a) 1 (b) 3

(c) 5 (d) 7

Multiple Choice Questions

7.30 Reliability of a software is dependent on number of errors

(a) removed
(b) remaining
(c) both (a) & (b)
(d) None of the above

7.31 Reliability of software is usually estimated at

(a) Analysis phase
(b) Design phase
(c) Coding phase
(d) Testing phase

7.32 CMM stands for

(a) Capacity maturity model
(b) Capability maturity model
(c) Cost management model
(d) Comprehensive maintenance model

7.33 Which level of CMM is for basic project management?

(a) Initial
(b) Repeatable
(c) Defined
(d) Managed

7.34 Which level of CMM is for process management?

(a) Initial
(b) Repeatable
(c) Defined
(d) Optimizing

Multiple Choice Questions

7.35 Which level of CMM is for process management?

(a) Initial
(b) Defined
(c) Managed
(d) Optimizing

7.36 CMM was developed at

(a) Harvard University
(b) Cambridge University
(c) Carnegie Mellon University
(d) Maryland University

7.37 McCall has developed a

(a) Quality model
(b) Process improvement model
(c) Requirement model
(d) Design model

7.38 The model to measure the software process improvement is called

(a) ISO 9000
(b) ISO 9126
(c) CMM
(d) Spiral model

7.39 The number of clauses used in ISO 9001 are

(a) 15
(b) 25
(c) 20
(d) 10

Multiple Choice Questions

7.40 ISO 9126 contains definitions of

(a) quality characteristics
(b) quality factors
(c) quality attributes
(d) All of the above

7.41 In ISO 9126, each characteristics is related to

(a) one attributes
(b) two attributes
(c) three attributes
(d) four attributes

7.42 In McCall quality model; product revision quality factor consist of

(a) Maintainability
(b) Flexibility
(c) Testability
(d) None of the above

7.43 Which is not a software reliability model ?

(a) The Jelinski-Moranda Model
(b) Basic execution time model
(c) Spiral model
(d) None of the above

7.44 Each maturity model is CMM has

(a) One KPA
(b) Equal KPAs
(c) Several KPAs
(d) no KPA

Multiple Choice Questions

7.45 KPA in CMM stands for

(a) Key Process Area
(b) Key Product Area
(c) Key Principal Area
(d) Key Performance Area

7.46 In reliability models, our emphasis is on

(a) errors
(b) faults
(c) failures
(d) bugs

7.47 Software does not break or wear out like hardware. What is your opinion?

(a) True
(b) False
(c) Can not say
(d) not fixed

7.48 Software reliability is defined with respect to

(a) time
(b) speed
(c) quality
(d) None of the above

7.49 MTTF stands for

(a) Mean time to failure
(b) Maximum time to failure
(c) Minimum time to failure
(d) None of the above

Multiple Choice Questions

7.50 ISO 9000 is a series of standards for quality management systems and has

(a) 2 related standards
(b) 5 related standards
(c) 10 related standards
(d) 25 related standards

Exercises

7.1 What is software reliability? Does it exist?

7.2 Explain the significance of bath tube curve of reliability with the help of a diagram.

7.3 Compare hardware reliability with software reliability.

7.4 What is software failure? How is it related with a fault?

7.5 Discuss the various ways of characterising failure occurrences with respect to time.

7.6 Describe the following terms:

(i) Operational profile (ii) Input space

(iii) MTBF (iv) MTTF

(v) Failure intensity.

Exercises

7.7 What are uses of reliability studies? How can one use software reliability measures to monitor the operational performance of software?

7.8 What is software quality? Discuss software quality attributes.

7.9 What do you mean by software quality standards? Illustrate their essence as well as benefits.

7.10 Describe the McCall software quality model. How many product quality factors are defined and why?

7.11 Discuss the relationship between quality factors and quality criteria in McCall's software quality model.

7.12 Explain the Boehm software quality model with the help of a block diagram.

7.13 What is ISO9126 ? What are the quality characteristics and attributes?

Exercises

7.14 Compare the ISO9126 with McCall software quality model and highlight few advantages of ISO9126.

7.15 Discuss the basic model of software reliability. How $\Delta\mu$ *and* $\Delta\tau$ can be calculated.

7.16 Assume that the initial failure intensity is 6 failures/CPU hr. The failure intensity decay parameter is 0.02/failure. We assume that 45 failures have been experienced. Calculate the current failure intensity.

7.17 Explain the basic & logarithmic Poisson model and their significance in reliability studies.

Exercises

7.18 Assume that a program will experience 150 failures in infinite time. It has now experienced 80. The initial failure intensity was 10 failures/CPU hr.

(i) Determine the current failure intensity

(ii) Calculate the failures experienced and failure intensity after 25 and 40 CPU hrs. of execution.

(iii) Compute additional failures and additional execution time required to reach the failure intensity objective of 2 failures/CPU hr.

Use the basic execution time model for the above mentioned calculations.

7.19 Write a short note on Logarithmic Poisson Execution time model. How can we calculate $\Delta\mu$ & $\Delta\tau$?

7.20 Assume that the initial failure intensity is 10 failures/CPU hr. The failure intensity decay parameter is 0.03/failure. We have experienced 75 failures upto this time. Find the failures experienced and failure intensity after 25 and 50 CPU hrs. of execution.

Exercises

7.21 The following parameters for basic and logarithmic Poisson models are given:

Basic execution time model	*Logarithmic Poisson execution time model*
λ_0 = 5 failures/CPU hr	λ_0 = 25 failures/CPU hr
V_0 = 125 failures	θ = 0.3/failure

Determine the additional failures and additional execution time required to reach the failure intensity objective of 0.1 failure/CPU hr. for both models.

7.22 Quality and reliability are related concepts but are fundamentally different in a number of ways. Discuss them.

7.23 Discuss the calendar time component model. Establish the relationship between calendar time to execution time.

Exercises

7.24 A program is expected to have 250 faults. It is also assumed that one fault may lead to one failure. The initial failure intensity is 5 failure/CPU hr. The program is released with a failure intensity objective of 4 failures/10 CPU hr. Calculate the number of failures experienced before release.

7.25 Explain the Jelinski-Moranda model of reliability theory. What is the relation between 't' and $'\lambda'$?

7.26 Describe the Mill's bug seeding model. Discuss few advantages of this model over other reliability models.

7.27 Explain how the CMM encourages continuous improvement of the software process.

7.28 Discuss various key process areas of CMM at various maturity levels.

7.29 Construct a table that correlates key process areas (KPAs) in the CMM with ISO9000.

7.30 Discuss the 20 clauses of ISO9001 and compare with the practices in the CMM.

Exercises

7.31 List the difference of CMM and ISO9001. Why is it suggested that CMM is the better choice than ISO9001?

7.32 Explain the significance of software reliability engineering. Discuss the advantage of using any software standard for software development?

7.33 What are the various key process areas at defined level in CMM? Describe activities associated with one key process area.

7.34 Discuss main requirements of ISO9001 and compare it with SEI capability maturity model.

7.35 Discuss the relative merits of ISO9001 certification and the SEI CMM based evaluation. Point out some of the shortcomings of the ISO9001 certification process as applied to the software industry.