Software Reliability

1. Hardware and Software Reliability:

As the demand for more large and complex systems has increased, the reliability of such systems has become a major concern for computer manufacturers and programmers.

A computer system consists of two major components: hardware and software. Although extensive research has been carried out on hardware reliability, the growing importance of recent software in complex applications dictates that the major focus has shifted to system software reliability and cost analysis. Software reliability is different from hardware reliability in the sense that software does not wear out or burn out. The software itself does not fail unless flaws within the software result in a failure in its dependent system.
Both hardware and software are required for the system to work and both have different phases through their operational life.

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

1. In **burn-in phase,** failure rate is quite high initially, and it starts decreasing gradually as the time progresses.
2. During **useful life phase or period**, failure rate is approximately constant.
3. Failure rate increase in **wear-out phase** due to wearing out/aging of components.

The best period is useful life period. The shape of the curve for failures is given in Fig. 1.

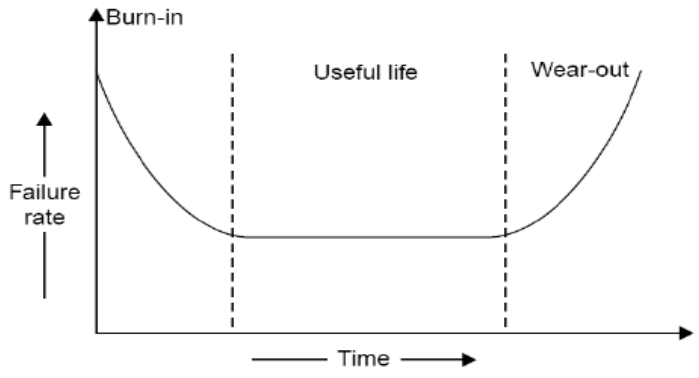


Fig. 1: Bath tub curve of hardware reliability.

The **software** component also has three life phases which are testing phase, useful life & obsolescence:

1. In the **testing phase,** failure rate is quite high initially, and it starts decreasing gradually as the time progresses.
2. During **useful life period**, failure rate is approximately constant.
3. Failure rate does not increase in **obsolescence** but the software itself is no longer needed (different better software is developed).

The best period for the software component is also useful life period. The software reliability curve is shown in Fig. 2.

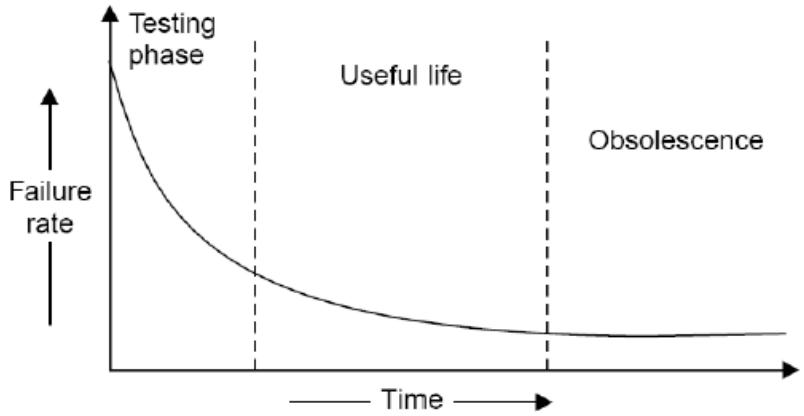


Fig. 2: Software reliability curve (failure rate versus time)

There are many factors that lead to a software component becoming obsolete (retired), such as:

- **Technological obsolescence**: the software is no longer being sold by the original vendor, being unable to expand or renew licensing agreements.

- **Functional obsolescence**: The functionality of the software will become obsolete due to hardware requirements or other software changes to the system. Software vendors produce new software to run on improved hardware and make older software versions obsolete. Newer versions of software can also make other software obsolete.

- **Logistical Obsolescence**: Access to software is limited or terminated due to digital media obsolescence, formatting, or degradation.

1.2. Defining Software Reliability:

Software Reliability can be defined in many ways, all reflecting the same idea:

- " *the ability of a system or component to perform its required functions under stated conditions for a specified period of time*."

or

- *"the probability of a failure-free operation of a program for a specified time in a specified environment."*

Software Reliability Engineering (SRE) is defined as *a set of engineering principals (techniques) that emphasizes dependability in the lifecycle management of a product.*

Research on software reliability engineering has been conducted during the past three decades and numerous statistical models have been proposed for estimating software reliability. Most existing models for predicting software reliability are based purely on the observation of software product failures where they require a considerable amount of failure data to obtain an accurate reliability prediction. Some other research efforts recently have developed reliability models addressing fault coverage, testing coverage, and imperfect debugging processes.

Many researchers are currently pursuing the development of statistical models, based on nonhomogeneous Poisson process, semi quasi renewal, time series, that can be used to evaluate the reliability of real-world software systems.

1.3 Failure Occurrences:

First, it is important to define terms such as "error", "fault" and "failure":

- An **error** is a mental mistake made by the programmer or designer.
- A **fault** is the manifestation of that error in the code.
- A software **failure** is defined as the occurrence of an incorrect output as a result of an input value that is received with respect to the specification.

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

Failure behavior is affected by two principal factors:

1. The number of faults in the software being executed.
2. The execution environment or the operational profile of execution.

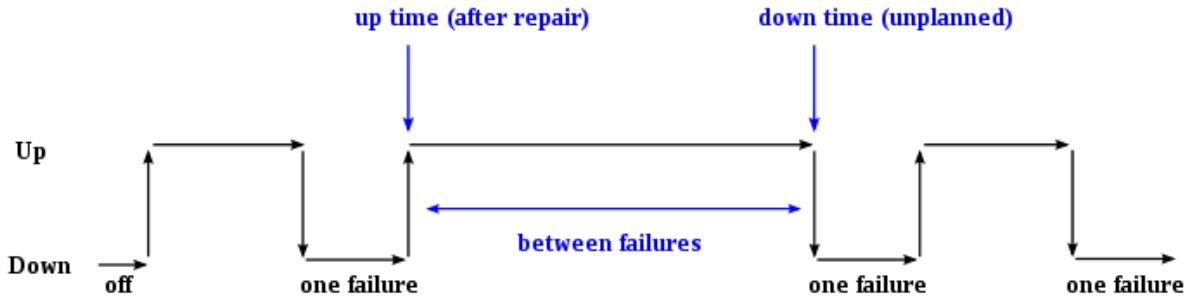
Time:

Reliability quantities are defined with respect to time, although it would be possible to define them with respect to other variables. We are concerned with three kinds of time:

- the **execution time** for a software is the CPU time that is actually spent by the computer in executing the software;
- the **calendar time** is the time people normally experience in terms of years, months, weeks, days, etc.;
- and the **clock time** is the elapsed time from start to end of computer execution in running the software.

There are four general ways of characterizing 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.



Time Between Failures = { down time - up time}

1.4 Causes of Failures

1. Poor Design, Production and Use
2. System Complexity
3. Poor Maintenance
4. Communication and Coordination
5. Human Reliability

1.5 Reliability Measures

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

1. They can be used to evaluate software engineering technology quantitatively.
2. They offer you the possibility of evaluating development status during the test phases of a project.
3. One can use them 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. then we are more capable of making informed decisions.

1.6 Software Quality Attributes

There are four main attribute domains when software quality is involved:

1. Reliability
2. Usability
3. Maintainability
4. Adaptability

Each of the attribute domains has its own attributes which are used to measure its intended purpose. Figs.3 & 4 shows the four attribute domains with their respected attributes.

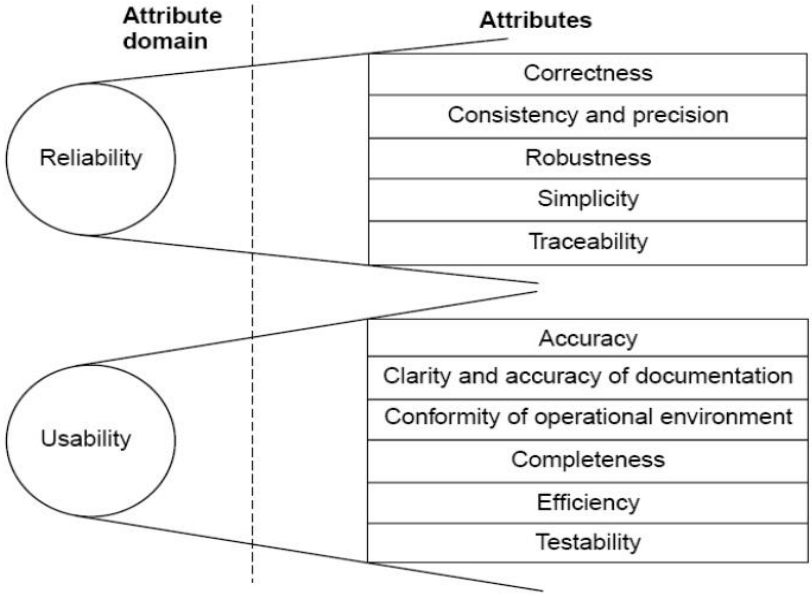


Fig.3: Software quality attributes

1.6.1 Reliability:

1) **Reliability:** The extent to which a software performs its intended functions without failure. (Fig. 3)
   a) **Correctness:** The extent to which a software meets its specifications.
   b) **Consistency & precision:** The extent to which a software is consistent and give results with precision.
   c) **Robustness:** The extent to which a software tolerates the unexpected problems.
   d) **Simplicity:** The extent to which a software is simple in its operations.
   e) **Traceability:** The extent to which an error is traceable in order to fix it.

1.6.2 Usability

1) **Usability:** The extent of effort required to learn, operate and understand the functions of the software (Fig. 3)
   a) **Accuracy:** Meeting specifications with precision.
   b) **Clarity & Accuracy of documentation:** The extent to which documents are clearly & accurately written.
   c) **Conformity of operational environment:** The extent to which a software is in conformity of operational environment.
   d) **Completeness:** The extent to which a software has specified functions.
   e) **Efficiency:** The amount of computing resources and code required by software to perform a function.
   f) **Testability:** The effort required to test a software to ensure that it performs its intended functions.

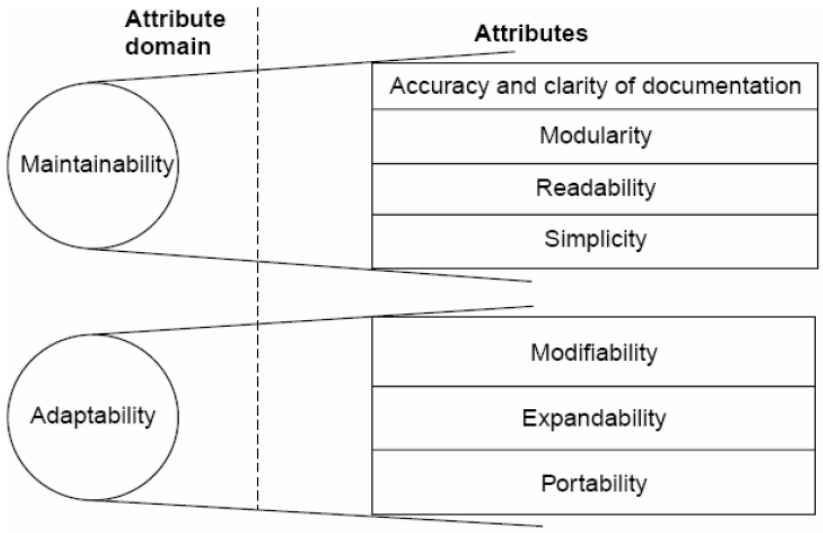


Fig.4 : Software quality attributes

1.6.3 Maintainability

1) **Maintainability:** The effort required to locate and fix an error during maintenance phase. (Fig.4)
   a) **Accuracy & Clarity of documentation:** The extent to which documents are clearly & accurately written.
   b) **Modularity:** It is the extent of ease to implement, test, debug and maintain the software.
   c) **Readability:** The extent to which a software is readable in order to understand.
   d) **Simplicity:** The extent to which a software is simple in its operations.

1.6.4 Adaptability

1) **Adaptability:** The extent to which a software is adaptable to new platforms & technologies. (Fig. 4)
   a) **Modifiability:** The effort required to modify a software during maintenance phase.
   b) **Expandability:** The extent to which a software is expandable without undesirable side effects.
   c) **Portability:** The effort required to transfer a program from one platform to another platform.

1.7 McCall Software Quality Model

McCall model is a software quality measuring model that uses different quality factors which incorporate the quality attributes stated earlier. The model operates under three main phases:

1) Product operation
2) Product revision
3) Product transition

All of the model phases have their own quality factors, as shown in Fig. 5

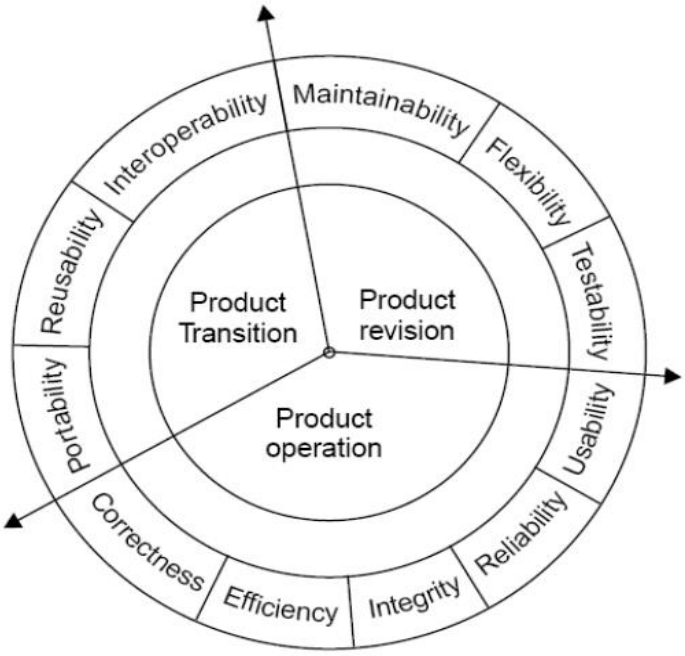


Fig.5: Software quality factors

1.7.1 Product Operation

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

1. **Correctness**
2. **Efficiency**
3. **Integrity:** The extent to which access to software or data by the unauthorized persons can be controlled.
4. **Reliability**
5. **Usability**

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

1.7.2 Product Revision

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

1. **Maintainability**
2. **Flexibility:** The effort required to modify an operational program.
3. **Testability**

These factors are related to the testing & maintainability of software. They give us an idea about (a) ease of maintenance, (b) flexibility and (c) testing effort. Hence, they are combined under the umbrella of product revision.

1.7.3 Product Transition

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

1. **Portability**
2. **Reusability**
3. **Interoperability:** The effort required to couple one system with another.

1.8 Software Reliability Engineering (SRE)

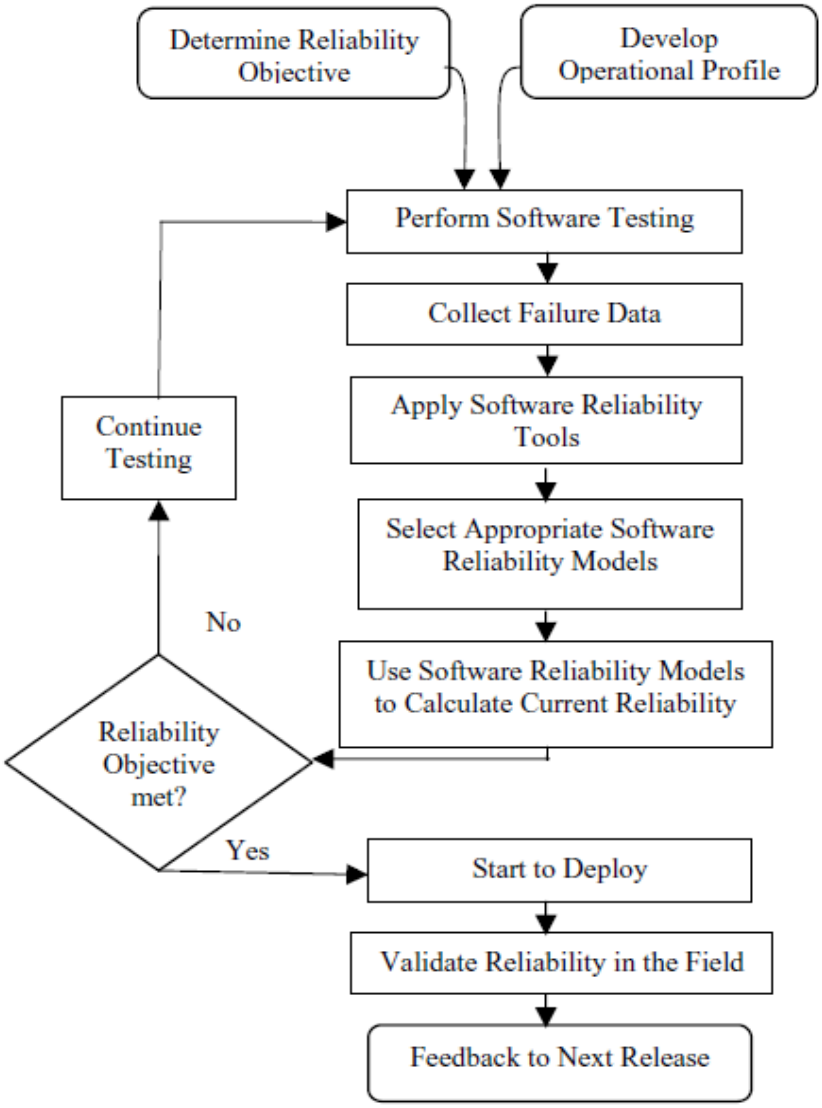


Fig.6: Software Reliability Engineering Process Overview

An SRE framework is shown in Fig. 6. First, a reliability objective is determined from the customer's viewpoint to maximize customer satisfaction, and customer usage is defined by developing an operational profile.

The software is then tested according to the operational profile and failure data collected, reliability is tracked during testing to determine the product release time. This activity may be repeated until a certain reliability level has been achieved.

1.8.1 Software Reliability Engineering Models

There are different and diverse models that can be used for SRE. Almost all models follow the same general principal, data will be entered to the model, a prediction and an estimation will exit it. Fig. 7 shows the general idea behind SRE models.

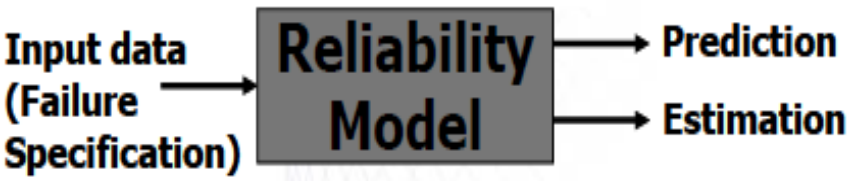


Fig.7: The General Principal behind SRE Models

1.8.2 The basic features of an SRE model

There are some basic features that all SRE models should possess. Mainly, it is important that the model is capable of:

1. Predicting probability of failure of a component or system.
2. Estimating the mean time to the next failure.
3. Predicting number of (remaining) failures during the development.

1.8.3 How to use a SRE model

There are some steps that must be followed while using any SRE model. These steps are:

1. Collecting failure data (failure specification).
2. Examining data.
3. Selecting a model.

4. Estimating model parameters.
5. Customizing the model using the estimated parameters.
6. Goodness-of-fit test.
7. Making reliability predictions.

1.8.4 SRE Model Types

There are different types of SRE models that can be used for different projects and in different ways based on many factors including the amount of data that is collected, the fault tolerance nature of the software project, etc…

There are various reliability growth models that have been derived from reliability experiments in a number of different application domains. Most of these models are exponential, with reliability increasing quickly as defects are discovered and removed. The increase then tails off and reaches a plateau as fewer and fewer defects are discovered and removed in the later stages of testing.

SRE models are generally categorized into **Equal Step Function Models** and **Random Step Function Models**, some examples of those models include:

1. Single failure model.
2. Reliability growth model.
3. Exponential failure class models.
4. Weibull and gamma failure class models.
5. Infinite failure category models.
6. Bayesian models.
7. Early life-cycle prediction models

The simplest model that illustrates the concept of reliability growth is a step function model. The reliability increases by a constant increment each time a fault (or a set of faults) is discovered and repaired (as shown in Fig. 8) and a new version of the software is created.

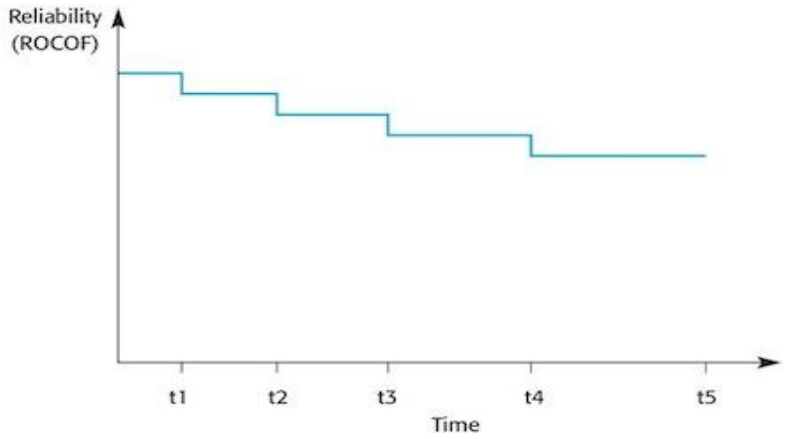


Fig.8: Equal-step function model of reliability growth

This model assumes that software repairs are always correctly implemented so that the number of software faults and associated failures decreases in each new version of the system. As repairs are made, the Rate Of oCcurrence Of software Failures (ROCOF) should therefore decrease, as shown in Fig. 8. Note that the time periods on the horizontal axis reflect the time between releases of the system for testing so they are normally of unequal length.

In practice, however, software faults are not always fixed during debugging and when you change a program, you sometimes introduce new faults into it. The probability of occurrence of these faults may be higher than the occurrence probability of the fault that has been repaired. Therefore, the system reliability may sometimes worsen in a new release rather than improve.

The simple equal-step reliability growth model also assumes that all faults contribute equally to reliability and that each fault repair contributes the same amount of reliability growth. However, not all faults are equally probable. Repairing the most common faults contributes more to reliability growth than does repairing faults that occur only occasionally. You are also likely to find these probable faults early in the testing process, so reliability may increase more than when later, less probable, faults are discovered.

Later models take these problems into account by introducing a random element into the reliability growth improvement effected by a software repair. Thus, each repair does not result in an equal amount of reliability improvement but varies depending on the random perturbation (Fig. 9).

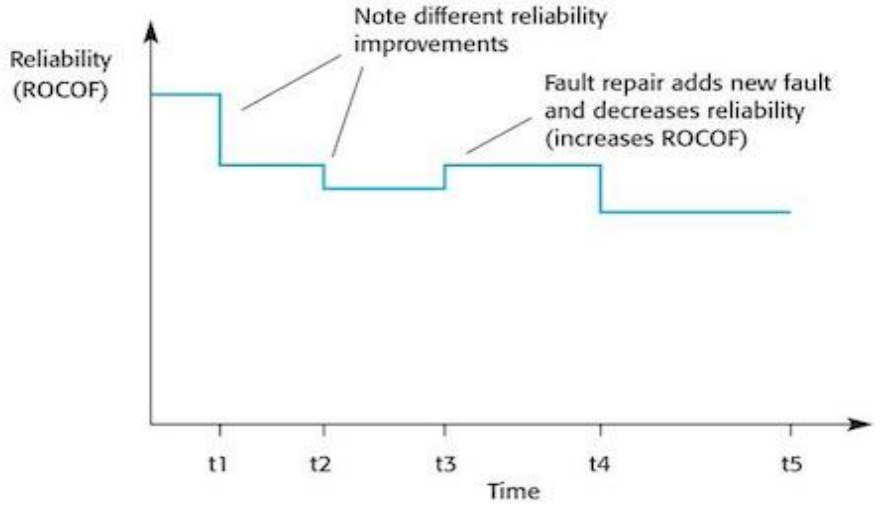


Fig.9: Random-step function model of reliability growth

Random-step models allow for negative reliability growth when a software repair introduces further errors. It also models the fact that as faults are repaired, the average improvement in reliability per repair decreases. The reason for this is that the most probable faults are likely to be discovered early in the testing process. Repairing these contributes most to reliability growth.

The above models are discrete models that reflect incremental reliability growth. When a new version of the software with repaired faults is delivered for testing it should have a lower rate of failure occurrence than the previous version. However, to predict the reliability that will be achieved after a given amount of testing continuous mathematical models are needed. Many models, derived from different application domains, have been created for this purpose.

2- System Reliability Analysis:

A **system** is a collection of components, subsystems and/or assemblies arranged to a specific design in order to achieve desired functions with acceptable performance and reliability.

Life data can be lifetimes of products in the marketplace, such as the time the product operated successfully or the time the product operated before it failed. The subsequent analysis and prediction are described as **life data analysis**.

In **life data analysis** and accelerated life testing data analysis, as well as other testing activities, one of the primary objectives is to obtain a **life distribution** that describes the times-to-failure of a component, subassembly, assembly or system. This analysis is based on the time of successful operation or time-to-failure data of the item (component), either under use conditions or from accelerated life tests.

As most products are made up of a number of components, the reliability of each component and the configuration of the system consisting of these components determines the **system reliability** (the reliability of the product).

The types of components, their quantities, their qualities and the manner in which they are arranged within the system have a direct effect on the system's reliability.

2-1 Basics of System Reliability Analysis

System Reliability Analysis is concerned with the construction of a model that represents the times to failure of the entire system based on the life distributions of the components, subassemblies and/or assemblies ("black boxes") from which it is composed, as illustrated in Fig. 10.

There are many specific reasons for looking at component data to estimate the overall system reliability. One of the most important is that in many situations it is easier and less expensive to test components/subsystems rather than entire systems.

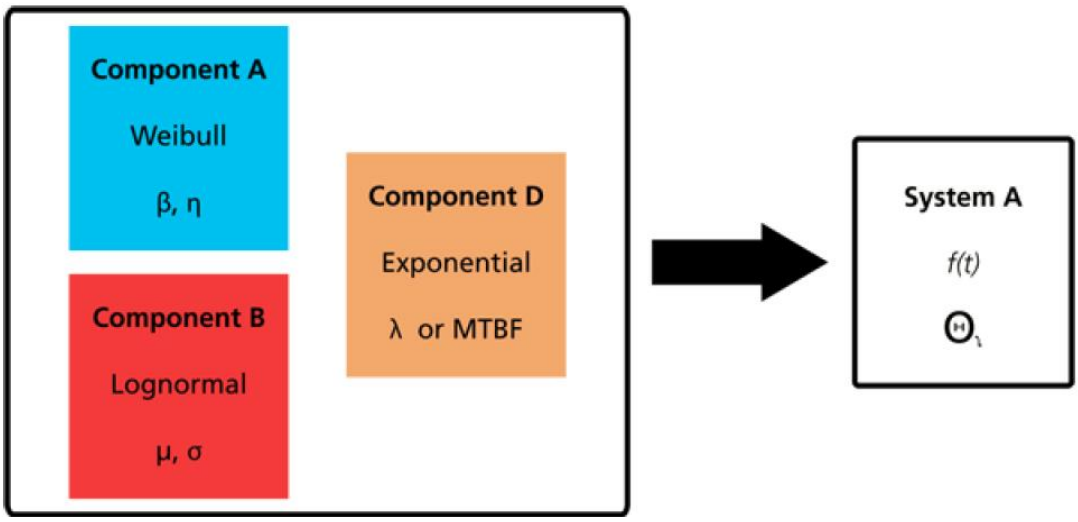


Fig. 10 Reliability Block Diagrams (RBDs)

2-2 Reliability Block Diagrams (RBDs)

A Reliability Block Diagram (RBD) is a diagrammatic method for showing how Component Reliability contributes to the success or failure of a complex system.

Block diagrams are widely used in engineering and science and exist in many different forms. They can also be used to describe the interrelation between the components and to define the system. When used in this fashion, the block diagram is then referred to as a Reliability Block Diagram (RBD).

A Reliability Block Diagram is a graphical representation of the components of the system and how they are reliability wise related or connected.

RBDs are constructed out of blocks. The blocks are connected with direction lines that represent the reliability relationship between the blocks. They are drawn as a series of blocks connected in parallel or series configuration. Each block represents a component of the system with a failure rate. Parallel paths are redundant, meaning that all of the parallel paths must fail for the parallel network to fail. By contrast, any failure along a series path causes the entire series path to fail.

A block is usually represented in the diagram by a rectangle. In a reliability block diagram, such blocks represent the component, subsystem or assembly at its chosen black box level (Fig. 11)

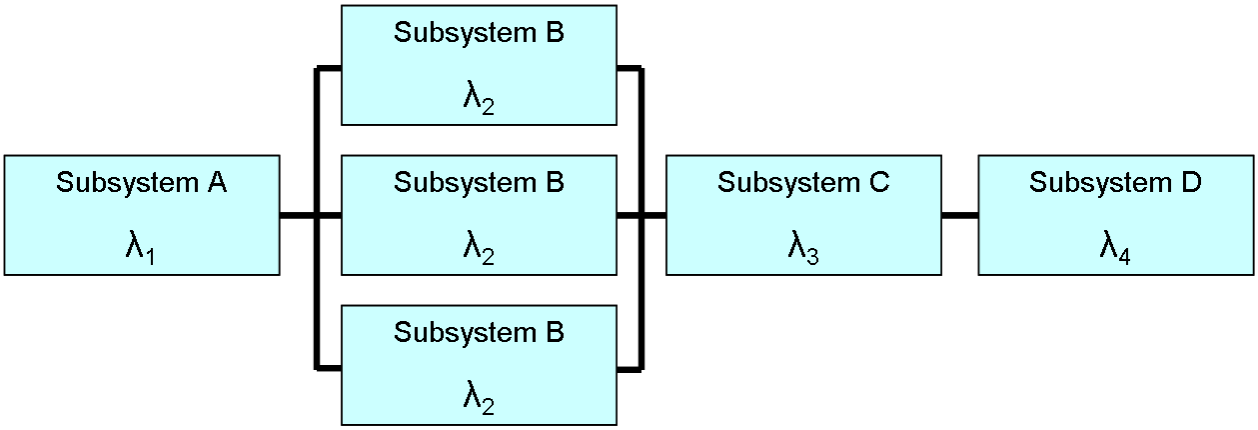


Fig. 11 A Reliability Block Diagram

It is possible for each block in a particular RBD to be represented by its own reliability block diagram, depending on the level of detail in question. For example, in an RBD of a car, the top level blocks could represent the major systems of the car, as illustrated in Fig. 12.

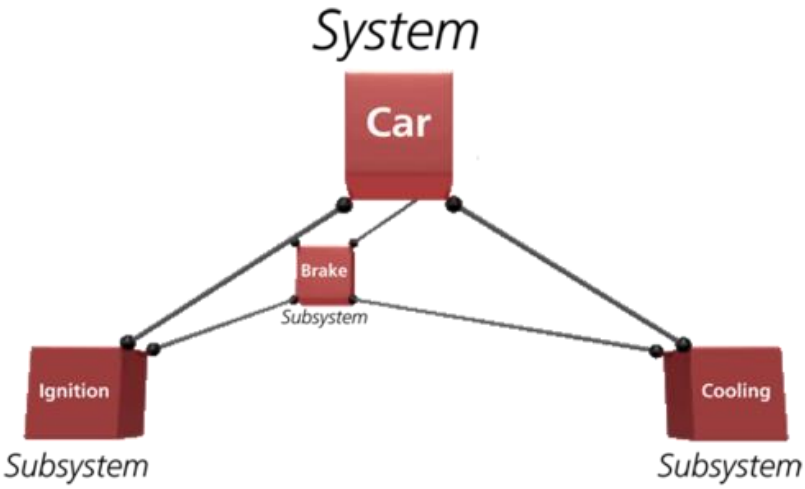


Fig. 12 Major Systems of the Car

Each of these systems could have its own RBDs in which the blocks represent the subsystems of that particular system. This could continue down through many levels of detail, all the way down to the level of the most basic components.

After defining the properties of each block in a system, the blocks can then be connected in a reliability-wise manner to create a reliability block diagram for the system.

The RBD provides a visual representation of the way the blocks are reliability-wise arranged. This means that a diagram will be created that represents the functioning state (success or failure) of the system in terms of the functioning states of its components. In other words, this diagram demonstrates the effect of the success or failure of a component on the success or failure of the system.

For example, if all components in a system must succeed in order for the system to succeed, the components will be arranged reliability-wise in series. If one of two components must succeed in order for the system to succeed, those two components will be arranged reliability-wise in parallel. RBDs and Analytical System Reliability discusses RBDs and diagramming methods.

2-3 Block Failure Models

Having segmented a product or process into parts, the first step in evaluating the reliability of a system is to obtain life/event data concerning each component/subsystem (each block). This information will allow the **Reliability Engineer** to characterize the life distribution of each component. Data can be obtained from different sources, including:

1. In-house reliability tests
2. Accelerated life tests
3. Field data
4. Warranty data
5. Engineering knowledge
6. Similarity to prior designs
7. Other reference sources

An overall system reliability prediction can be made by looking at the reliabilities of the components that make up the whole system or product. Here we will examine the methods of performing such calculations. The reliability-wise configuration of components must be determined beforehand. For this reason, we will first look at different component/subsystem configurations.

2-4 Component Configurations:

In order to construct a reliability block diagram, the reliability-wise configuration of the components must be determined. Consequently, the analysis method used for computing the reliability of a system will also depend on the reliability-wise configuration of the components/subsystems.

That configuration can be as simple as units arranged in a pure series or parallel configuration. There can also be systems of combined series/parallel configurations or complex systems that cannot be decomposed into groups of series and parallel configurations. The configuration types considered in this reference include:

- Series configuration.
- Simple parallel configuration.
- Combined (series and parallel) configuration.
- Complex configuration.
- k-out-of-n parallel configuration.
- Configuration with a load sharing container (presented in Load Sharing).
- Configuration with a standby container (presented in Standby Components).
- Configuration with inherited subdiagrams.
- Configuration with multi blocks.
- Configuration with mirrored blocks.

2-4-1 Series Systems

In a series configuration, a failure of any component results in the failure of the entire system.

In most cases, when considering complete systems at their basic subsystem level, it is found that these are arranged reliability-wise in a series configuration. For example, a personal computer may consist of four basic subsystems: the motherboard, the hard drive, the power supply and the processor. These are reliability-wise in series and a failure of any of these subsystems will cause a system failure.

The reliability of the system is the probability that unit 1 succeeds and unit 2 succeeds and all of the other units in the system succeed. So all n units must succeed for the system to succeed. The reliability of the system is then given by:

$$\begin{aligned} R_S =& P(X_1 \cap X_2 \cap ... \cap X_n) \\ =& P(X_1)P(X_2|X_1)P(X_3|X_1X_2) \cdots P(X_n|X_1X_2...X_{n-1}) \end{aligned}$$

where:

R_s : is the reliability of the system

X_i : is the event of unit *i* being operational

$P(X_i)$: is probability that unit *i* is operational

In the case where the failure of a component affects the failure rates of other components (i.e., the life distribution characteristics of the other components change when one component fails), then the conditional probabilities in equation above must be considered.

However, in the case of independent components, equation above becomes:

$$R_s = P(X_1)P(X_2)...P(X_n)$$

Or:

$$R_s = \prod_{i=1}^{n} P(X_i)$$

Or, in terms of individual component reliability:

$$R_s = \prod_{i=1}^{n} R_i$$

In other words, for a pure series system, the system reliability is equal to the product of the reliabilities of its constituent components.

Example: Calculating Reliability of a Series System

Three subsystems are reliability-wise in series and make up a system:

- subsystem 1 has a reliability of 99.5%,
- subsystem 2 has a reliability of 98.7%
- subsystem 3 has a reliability of 97.3%

for a mission of 100 hours. What is the overall reliability of the system for a 100-hour mission?

Sol:

Since the reliabilities of the subsystems are specified for 100 hours, the reliability of the system for a 100-hour mission is simply:

$$
\begin{aligned}
R_s &= R_1 \cdot R_2 \cdot R_3 \\
R_s &= 0.9950 \cdot 0.9870 \cdot 0.9730 \\
R_s &= 0.955549245 \\
R_s &= 95.55\%
\end{aligned}
$$

Effect of Component Reliability in Series Systems:

In a series configuration, the component with the least reliability has the biggest effect on the system's reliability. There is a saying that "*a chain is only as strong as its weakest link*." In a chain, all the rings are in series and if any of the rings break, the system fails. In addition, the weakest link in the chain is the one that will break first, The weakest link dictates the strength of the chain in the same way that the weakest component/subsystem dictates the reliability of a series system as in Fig. 13.

As a result, the reliability of a series system is always less than the reliability of the least reliable component.

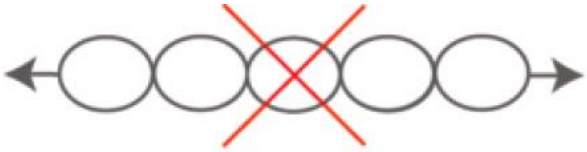


Fig. 13 Effect of Component Reliability in Series Systems

Example

Consider three components arranged reliability-wise in series, where, R1 = 70%, R2 = 80% and R3 = 90% (for a given time).

In Table (1), we can examine the effect of each component's reliability on the overall system reliability.

Table (1) System reliability for components

Component 1	Component 2	Component 3	System
0.7	0.8	0.9	0.504
0.8	0.8	0.9	0.576
0.7	0.9	0.9	0.567
0.7	0.8	0.99	0.555

The first row of the table shows the given reliability for each component and the corresponding system reliability for these values. In the second row, the reliability of Component 1 is increased by a value of 10% while keeping the reliabilities of the other two components constant. Similarly, by increasing the reliabilities of Components 2 and 3 in rows 3 and 4 by a value of 10%, while keeping the reliabilities of the other components at the given values, the effect of each component's reliability on the overall system reliability is shown.

It is clear that the highest value for the system's reliability was achieved when the reliability of Component 1, which is the least reliable component, was increased by a value of 10%. This conclusion can also be illustrated graphically, as shown in the Fig. 14.

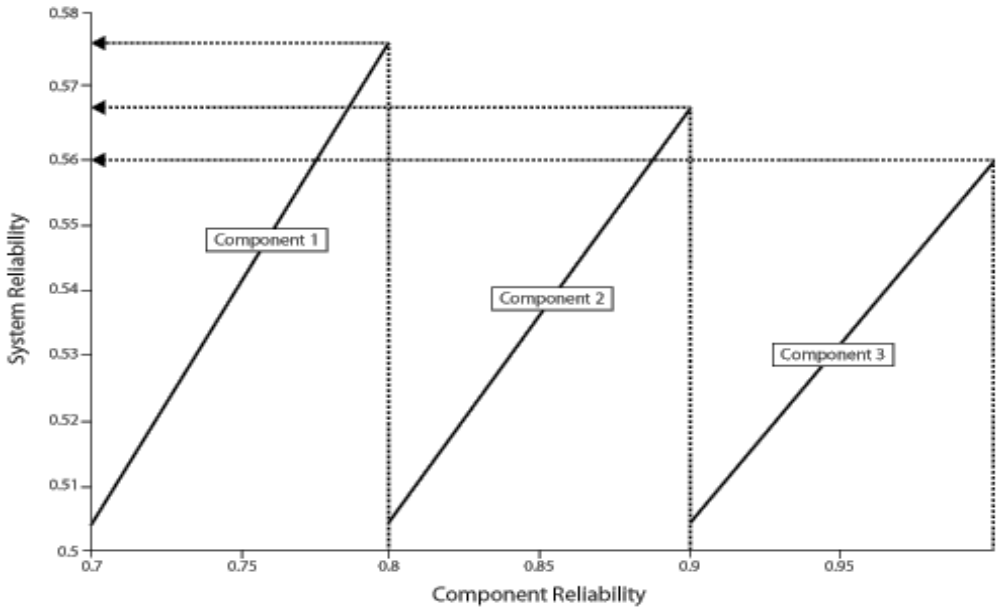


Fig. 14 System Reliability Vs. Component Reliability

The rate of change of the system's reliability with respect to each of the components is also plotted in Fig. 15. It can be seen that Component 1 has the steepest slope, which indicates that an increase in the reliability of Component 1 will result in a higher increase in the reliability of the system. In other words, Component 1 has a higher *reliability importance*.

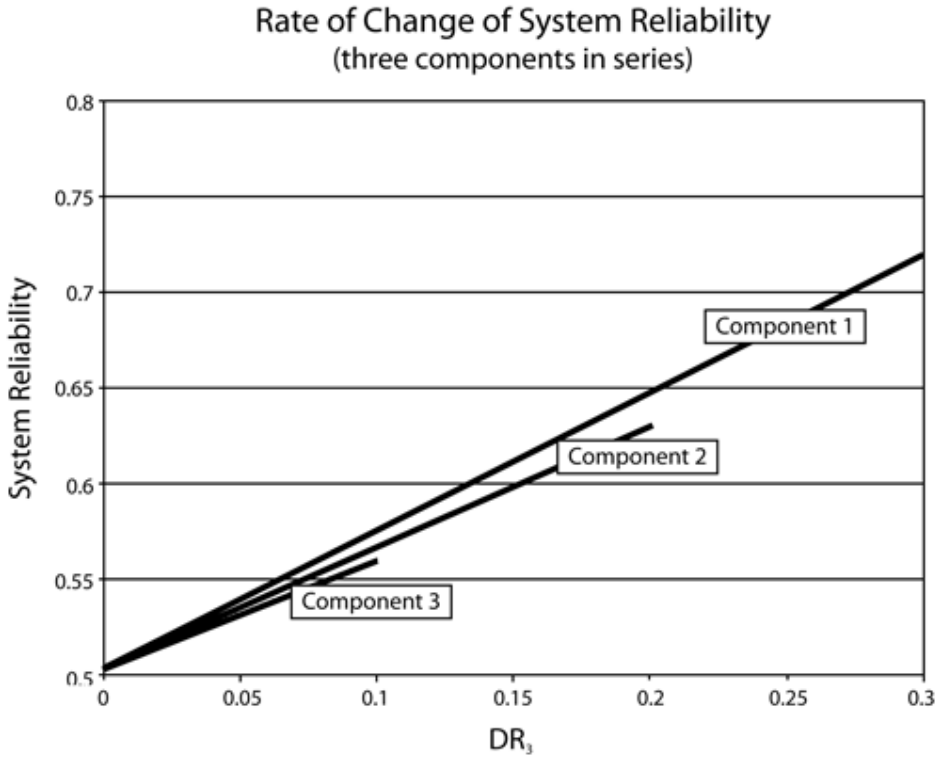


Fig. 15 Rate of change of the system's reliability with respect to each of the components

Effect of Number of Components in a Series System

The number of components is another concern in systems with components connected reliability-wise in series. As the number of components connected in series increases, the system's reliability decreases. Fig. 16 illustrates the effect of the number of components arranged reliability-wise in series on the system's reliability for different component reliability values.

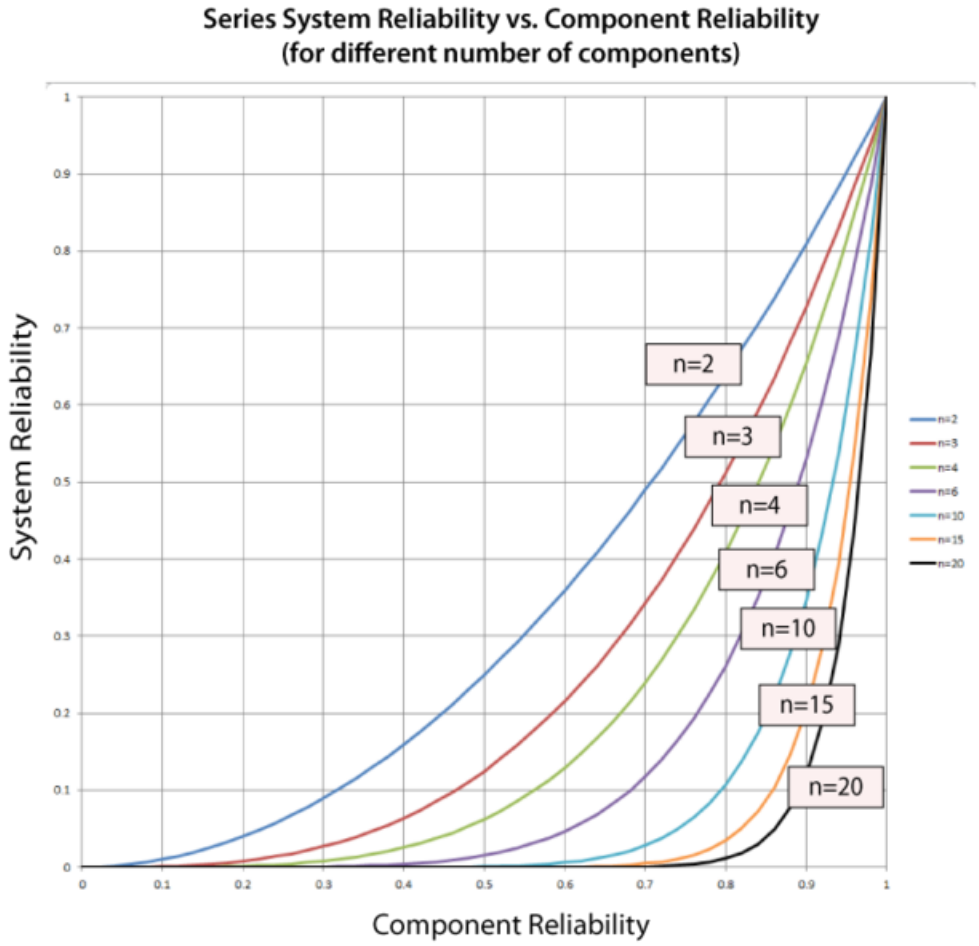


Fig. 16 The effect of the number of components arranged reliability-wise in series on the system's reliability

This figure also demonstrates the dramatic effect that the number of components has on the system's reliability, particularly when the component reliability is low. In other words, in order to achieve a high system reliability, the component reliability must be high also, especially for systems with many components arranged reliability-wise in series.

Example

Consider a system that consists of a single component. The reliability of the component is 95%, thus the reliability of the system is 95%. What would the reliability of the system be if there were more than one component (with the same individual reliability) in series?

Table (2) shows the effect on the system's reliability by adding consecutive components (with the same reliability) in series.

Table (2) System's reliability by adding consecutive components

Number of Components	System Reliability
1	0.95
2	0.9025
4	0.8145
6	0.7351
8	0.6634
10	0.5987

2-4-2 Parallel Systems

In a simple parallel system, as shown in Fig. 17, at least one of the units must succeed for the system to succeed. Units in parallel are also referred to as redundant units. Redundancy is a very important aspect of system design and reliability in that adding redundancy is one of several methods of improving system reliability.

Parallel systems are widely used in the aerospace industry and generally used in mission critical systems. Other example applications include the RAID computer hard drive systems, brake systems and support cables in bridges.

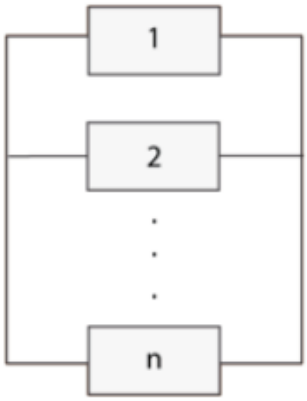


Fig. 17 Effect of Component Reliability in a Parallel Configuration

The probability of failure, or unreliability, for a system with n statistically independent parallel components is the probability that unit 1 fails and unit 2 fails and all of the other units in the system fail. So in a parallel system, all n units must fail for the system to fail. Put another way, if

unit 1 succeeds or unit 2 succeeds or any of the n units succeeds, then the system succeeds.

Reliability (probability of Success) = 1- probability of Failure

The probability of Failure for the system is then given by:

$$\begin{aligned} Q_s &= P(X_1 \cap X_2 \cap ... \cap X_n) \\ &= P(X_1)P(X_2|X_1)P(X_3|X_1X_2)...P(X_n|X_1X_2...X_{n-1}) \end{aligned}$$

where:

Q_s :is the unreliability of the system

X_i :is the event of failure of unit i

$P(X_i)$:is the probability of failure of unit i

In the case where the failure of a component affects the failure rates of other components, then the conditional probabilities in equation above must be considered. However, in the case of independent components, the equation above becomes:

$$Q_s = P(X_1)P(X_2)...P(X_n)$$

or:

$$Q_s = \prod_{i=1}^{n} P(X_i)$$

Or, in terms of component unreliability:

$$Q_s = \prod_{i=1}^{n} Q_i$$

In series systems, the system reliability was the product of the component reliabilities; whereas the parallel system has the overall system unreliability (probability of Failure) as the product of the component unreliabilities.

The reliability of the parallel system is then given by:

$$\begin{aligned} R_s =& 1 - Qs = 1 - (Q_1 \cdot Q_2 \cdot ... \cdot Q_n) \\ =& 1 - [(1 - R_1) \cdot (1 - R_2) \cdot ... \cdot (1 - R_n)] \\ =& 1 - \prod_{i=1}^{n} (1 - R_i) \end{aligned}$$

Example:

Consider a system consisting of three subsystems arranged reliability-wise in parallel.

Subsystem 1 has a reliability of 99.5%,

Subsystem 2 has a reliability of 98.7% and

Subsystem 3 has a reliability of 97.3%

for a mission of 100 hours.

What is the overall reliability of the system for a 100-hour mission?

Since the reliabilities of the subsystems are specified for 100 hours, the reliability of the system for a 100-hour mission is:

R_s=1- [(1-0.9950).(1-0.9870).(1-0.9730)]

= 1- 0.000001755

= 0.999998245

Effect of a Component's Reliability in a Parallel System:

When we examined a system of components in series, we found that the least reliable component has the biggest effect on the reliability of the system. However, the component with the highest reliability in a parallel configuration has the biggest effect on the system's reliability, since the most reliable component is the one that will most likely fail last. This is a very important property of the parallel configuration, specifically in the design and improvement of systems.

Example:

Consider three components arranged reliability-wise in parallel with

R1 = 60 %, R2 = 70 %, and R3 = 80 %.

The corresponding reliability for the system is R_s = 97.6% . In Table (3), we can examine the effect of each component's reliability on the overall system reliability. The first row of the table shows the given reliability for each

component and the corresponding system reliability for these values. In the second row, the reliability of Component 1 is increased by a value of 10% while keeping the reliabilities of the other two components constant. Similarly, by increasing the reliabilities of Components 2 and 3 in the third and fourth rows by a value of 10% while keeping the reliabilities of the other components at the given values, we can observe the effect of each component's reliability on the overall system reliability.

It is clear from Fig. 18 that the highest value for the system's reliability was achieved when the reliability of Component 3, which is the most reliable component, was increased. Once again, this is the opposite of what was encountered with a pure series system.

Table (3) System Reliability for Combinations of Components' Reliability

Component1	Component2	Component3	System Reliability
0.6	0.7	0.8	0.976
0.7	0.7	0.8	0.982
0.6	0.8	0.8	0.984
0.6	0.7	0.9	0.988

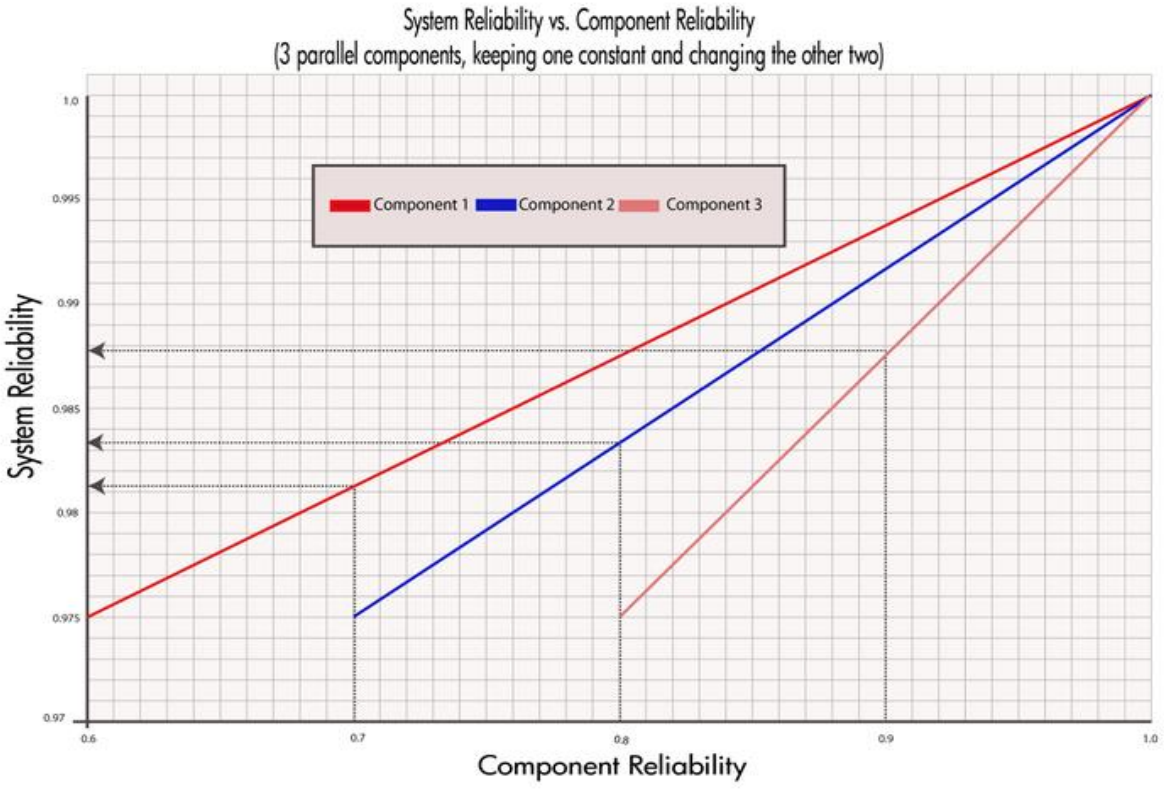


Fig. 18 Effect of a Component's Reliability in a Parallel System

Effect of Number of Components in a Parallel System

In the case of the parallel configuration, the number of components has the opposite effect of the one observed for the series configuration. For a parallel configuration, as the number of components/subsystems increases, the system's reliability increases. Fig. 19 illustrates that a high system reliability can be achieved with low reliability components, provided that there are a sufficient number of components in parallel.

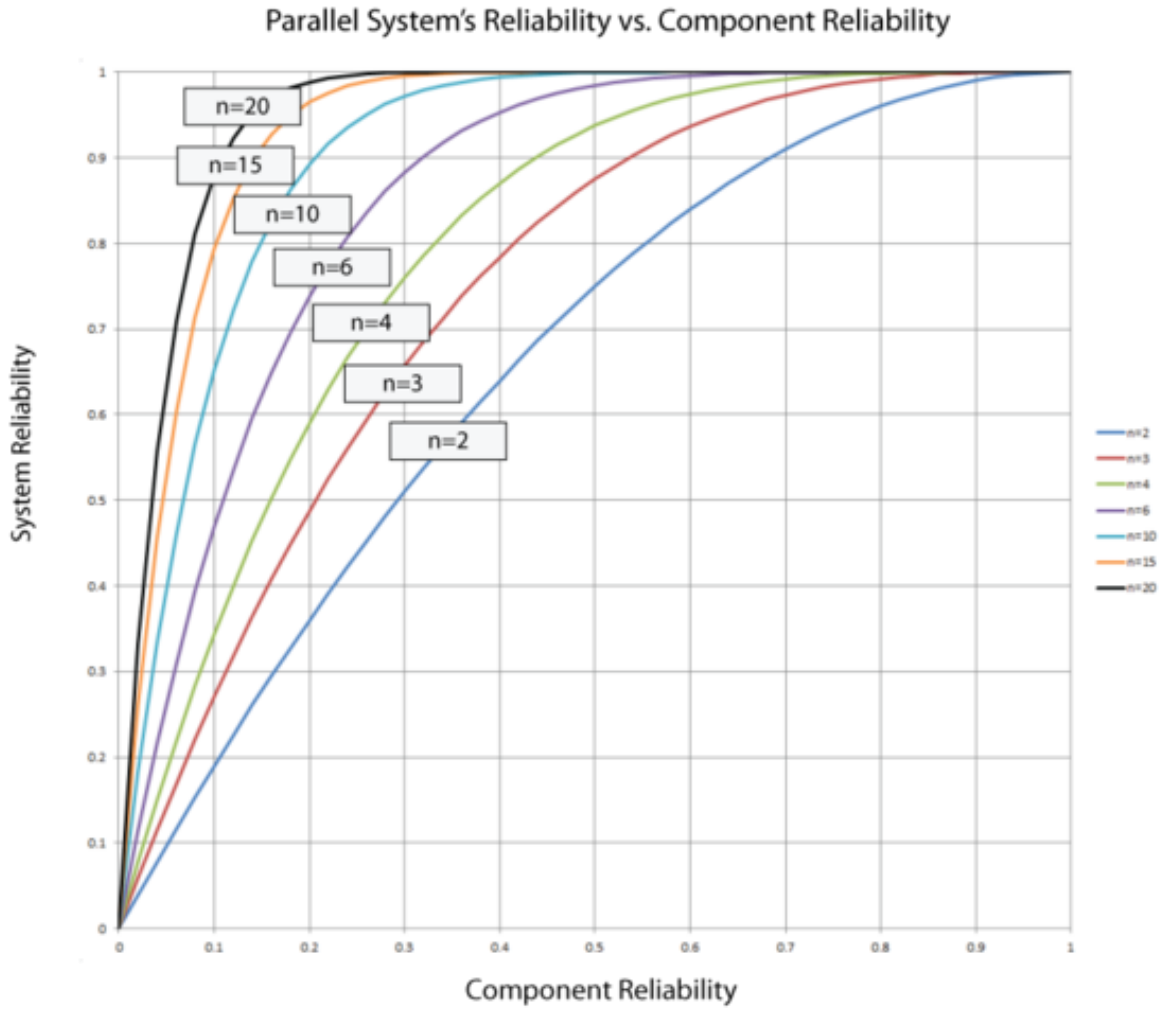


Fig. 19 Effect of Number of Components in a Parallel System

Example:

Consider a system that consists of a single component. The reliability of the component is 60%, thus the reliability of the system is 60%. What would the reliability of the system be if the system were composed of two, four or six such components in parallel?

Table (4) shows the system reliability with increasing components. Clearly, the reliability of a system can be improved by adding redundancy. However, it must be noted that doing so is usually costly in terms of additional components, additional weight, volume, etc.

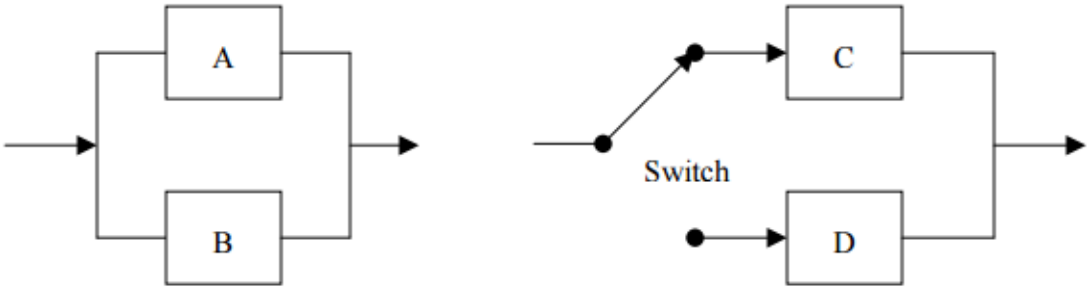
Table 4: System reliability as a function of the number of components.

Number of components	System Reliability
1	0.6
2	0.84
4	0.9744
6	0.9959

Active and Passive Redundancy

Redundancy is defined as the use of additional components or sub-systems beyond the number actually required for the system to operate reliably. A basic parallel system has inherent redundancy since the failure of one or more components does not result in a system failure as long as one component remains functional.

Redundancies can be categorized as active or passive (standby) redundancy. In systems with active redundancy all redundant components are in operation and are sharing the load with the main component. Upon failure of one component, the surviving components carry the load, and as a result, the failure rate of the surviving components may be increased (Fig 20 (a)). The redundant or back-up components in passive or standby systems start operating only when one or more fail. The back-up components remain dormant until needed (Fig. 20 (b)).



(a) **(b)**

Fig. 20

Assuming R(A) = R(B) = R(C) = R(D) and a perfect switch in block diagram (b) it seems that active and passive redundant systems would have the same reliability. However, whereas components A, B, and C start operating at time t = 0, component D does not start to operate until component C fails. Hence, all things equal, the passive redundant system depicted in (b) should have a higher reliability.

Passive (standby) redundant systems can fall into two categories:

A) **Hot standby** The standby components have the same failure rate as the primary component. The failure rate of one component is not affected by the performance/non-performance of the other components. Hence, the components are statistically independent.

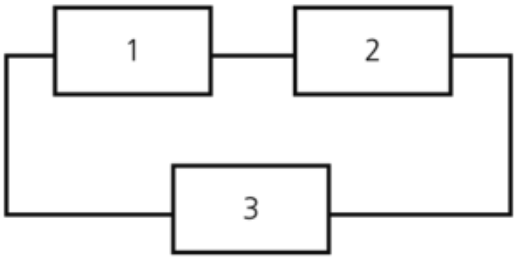
B) **Cold standby** The standby components have a zero failure rate. They do not fail when they are in standby mode. If and when the primary component fails, a standby component becomes the primary component with a non-zero failure rate.

2-4-3 Combination of Series and Parallel

While many smaller systems can be accurately represented by either a simple series or parallel configuration, there may be larger systems that involve both series and parallel configurations in the overall system. Such systems can be analyzed by calculating the reliabilities for the individual series and parallel sections and then combining them in the appropriate manner. Such a methodology is illustrated in the following example.

Example:

Consider a system with three components. Units 1 and 2 are connected in series and Unit 3 is connected in parallel with the first two, as shown in the next figure.



What is the reliability of the system if R1=99.5% , R2= 98.7% and R3=97.3% at 100 hours?

First, the reliability of the series segment consisting of Units 1 and 2 is calculated:

$$R_{1,2} = R_1 \cdot R_2$$
$$R_{1,2} = 0.9950 \cdot 0.9870$$
$$R_{1,2} = 0.982065 \text{ or } 98.2065\%$$

The reliability of the overall system is then calculated by treating Units 1 and 2 as one unit with a reliability of 98.2065% connected in parallel with Unit 3. Therefore:

$$R_s = 1 - [(1 - 0.982065) \cdot (1 - 0.973000)]$$
$$R_s = 1 - 0.000484245$$
$$R_s = 0.999515755$$
$$R_s = 99.95\%$$

3- BASIC RELIABILITY METRICS

Reliability metrics are used to quantitatively express the reliability of the software product. The choice of which metric is to be used depends upon the type of system to which it applies & the requirements of the application domain. Measuring the software reliability is a difficult problem because we don't have a good understanding about the nature of software. It is difficult to find a suitable way to measure software reliability, and most of the aspects related to software reliability. Even the software sizes have no uniform definition.

Some reliability metrics which can be used to quantify the reliability of the software product are given in the following subsections.

3-1 System Mean Time To Failure (MTTF)

MTTF is defined as the time interval between the successive failures. An MTTF of 200 means that one failure can be expected every 200 time units. The time units are totally dependent on the system and it can even be specified in the number of transactions. MTTF is relevant for systems with long transactions. For example, it is suitable for computer aided design systems where a designer will work on a design for several hours as well as for Word-processor systems.

Although the MTTF measure is one of the most widely used reliability calculations, it is also one of the most misused calculations. It has been misinterpreted as “guaranteed minimum lifetime”. Consider the results as given in Table (5) for a twelve-component life duration test.

Table (5) Results of a twelve-component life duration test

Component	Time to failure (hours)
1	4510
2	3690
3	3550
4	5280
5	2595
6	3690
7	920
8	3890
9	4320
10	4770
11	3955
12	2750
MTTF	3660

$$\text{MTTF} = \frac{\text{Total hours of operation}}{\text{Total number of items}}$$

Using a basic averaging technique, the component MTTF of 3660 hours was estimated. However, one of the components failed after 920 hours. Therefore, it is important to note that the system MTTF denotes the average time to failure.

3-2 Mean Time To Repair (MTTR)

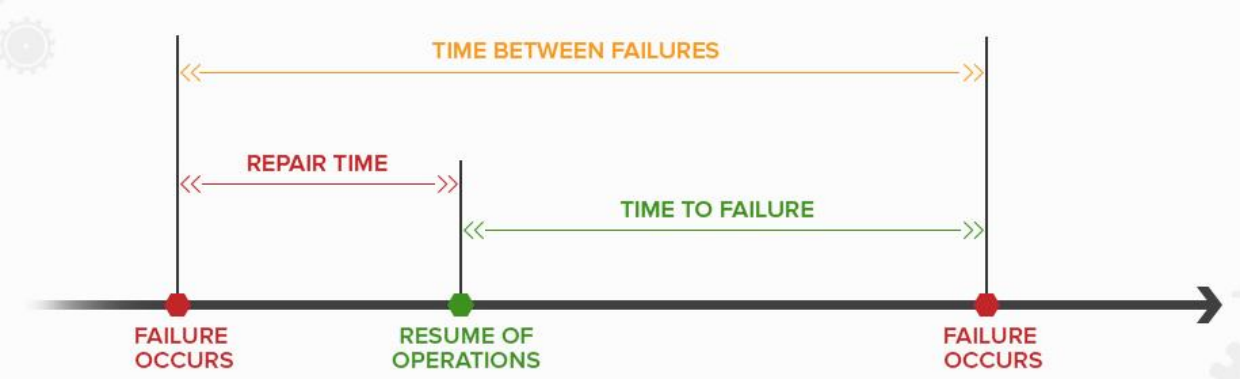
Once the failure occur sometime is required to fix the error. MTTR measures the average time it takes to track the errors causing the failure & to fix them.

$$\text{MTTR} = \frac{\text{Total maintenance time}}{\text{Total Number of Repairs}}$$

3-3 Mean Time Between Failures (MTBF)

We can combine Mean Time to Failure **(**MTTF) & Mean Time To Repair (MTTR) metrics to get the MTBF metric.

MTBF = MTTR + MTTF



Thus, an MTBF of 300 indicates that once the failure occurs, the next failure is expected to occur only after 300 hours. In this case the time measurements are real time & not the execution time as in MTTF.

3-4 Rate Of oCcurrence Of Failure (ROCOF)

It is the number of failures occurring in unit time interval. The number of unexpected events over a particular time of operation. ROCOF is the frequency of occurrence with which unexpected behavior is likely to occur. An ROCOF of 0.02 means that two failures are likely to occur in each 100 operational time unit steps. It is also called failure intensity metric.

3-5 Probability Of Failure On Demand (POFOD)

POFOD is defined as the probability that the system will fail when a service is requested. It is the number of system failures given a number of systems inputs. POFOD is the likelihood that the system will fail when a service request is made. A POFOD of 0.1 means that one out of a ten service requests may result in failure. POFOD is an important measure for safety critical systems. POFOD is appropriate for protection systems where services are demanded occasionally.

3-6 Availability (AVAIL)

Availability is the probability that the system is available for use at a given time. It takes into account the repair time & the restart time for the system. An availability of 0.995 means that in every 1000 time units, the system is likely to be available for 995 of these. The percentage of time that a system is available for use, taking into account planned and unplanned downtime. If a system is down an average of four hours out of 100 hours of operation, its AVAIL is 96%.

The availability of a system is defined as the probability that the system is successful at time t. Mathematically,

$$\text{AVAIL} = \frac{\text{MTBF}}{\text{MTBF} + \text{Mean down time}}$$

Availability is a measure of success used primarily for repairable systems. For non-repairable systems, availability, A(t), equals reliability, R(t). In repairable systems, A(t) will be equal to or greater than R(t).

3-7 Maintainability

When a system fails to perform satisfactorily, repair is normally carried out to locate and correct the fault. The system is restored to operational effectiveness by making an adjustment or by replacing a component.

Maintainability is defined as the probability that a failed system will be restored to specified conditions within a given period of time when maintenance is performed according to prescribed procedures and resources.

In other words, maintainability is the probability of isolating and repairing a fault in a system within a given time. Maintainability engineers must work with system designers to ensure that the system product can be maintained by the customer efficiently and cost effectively. This function requires the analysis of part removal, replacement, tear-down, and build-up of the product in order to determine the required time to carry out the operation, the necessary skill, the type of support equipment and the documentation.

The system repair time consists of two separate intervals: passive repair time and active repair time. Passive repair time is mainly determined by the time taken by service engineers to travel to the customer site. In many cases, the cost of travel time exceeds the cost of the actual repair. Active repair time is directly affected by the system design and is listed as follows:

1. The time between the occurrence of a failure and the system user becoming aware that it has occurred.
2. The time needed to detect a fault and isolate the replaceable component(s).
3. The time needed to replace the faulty component(s).
4. The time needed to verify that the fault has been corrected and the system is fully operational.

The active repair time can be improved significantly by designing the system in such a way that faults may be quickly detected and isolated. As more complex systems are designed, it becomes more difficult to isolate the faults.

4- Software Metrics related to Reliability:

4-1 Product Metrics

Product metrics are those which are used to build the artifacts i.e. requirement specification documents, system design documents etc. These metrics help in assessment if the product is good enough through reports on attributes like usability, reliability, maintainability & portability. Here measurements are taken from the actual body of the source code.

- **Software size** is thought to be reflective of complexity, development effort and reliability. Lines of Code (LOC), or LOC in thousands (KLOC), is an intuitive initial approach to measuring software size. The basis of LOC is that program length can be used as a predictor of program characteristics such as effort &ease of maintenance. It is a measure of the functional complexity of the program and is independent of the programming language.

- **Function point** metric is a method to measure the functionality of a proposed software development based on the count of inputs, outputs, master files, inquires, and interfaces.

- **Test coverage** metric estimate fault and reliability by performing tests on software products, assuming that software reliability is a function of the portion of software that is successfully verified or tested.

- **Complexity** is directly related to software reliability, so representing complexity is important. Complexity-oriented metrics is a method of determining the complexity of a program's control structure, by simplifying the code into a graphical representation.

- **Quality** metrics measures the quality at various stages of software product development. An important quality metric is defect removal efficiency (DRE).

4-2 Project Management Metrics

Project metrics describe the project characteristics and execution. If there is good management of project by the programmer then this help us to achieve better products. Relationship exists between the development process and the ability to complete projects on time and within the desired quality objectives. Cost increase when developers use inadequate processes. Higher reliability can

be achieved by using better development process, risk management process, configuration management process.

These metrics tells about:-

- Number of software developers
- Staffing pattern over the life-cycle of the software
- Cost and schedule
- Productivity

4-3 Process Metrics

Process metrics quantify useful attributes of the software development process & its environment. They tell if the process is functioning optimally as they report on attributes like cycle time & rework time. The goal of process metric is to do the right job on first time through the process. The quality of the product is a direct function of the process. So process metrics can be used to estimate, monitor and improve the reliability and quality of software. 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
- Maturity of the process

4-4 Fault and Failure Metrics

A fault is a defect in a program which arises when programmer makes an error and causes failure when executed under particular conditions. These metrics are used to determine the failure-free execution software. To achieve this goal, number of faults found during testing and the failures or other problems which are reported by the user after delivery are collected, summarized and analyzed. Failure metrics are based upon customer information regarding failures found after release of the software. The failure data collected is therefore used to calculate failure density, Mean Time Between Failures (MTBF) or other parameters to measure or predict software reliability.

5- Software Reliability Tools

Several software reliability tools are available for users to apply one or more of the software reliability model to a development effort and to determine the applicability of a particular model to a set of failure data. A major issue in modeling software reliability lies in the ease-of-use of currently available tools.

The following tasks are handled by the SRE tools:
- Collecting failure and test time information
- Calculating estimates of model parameters using information available.
- Testing to fit a model against the collected information.
- Selecting a model to make predictions of remaining faults, time to test, etc.
- Applying the model

Researchers developed various tools for estimating Software Reliability. These tools give the detail analysis of reliability in such a way that it can be further utilized to improve the reliability criteria for real-time applications . Any Software Reliability Estimation Tool involves 5 main steps:

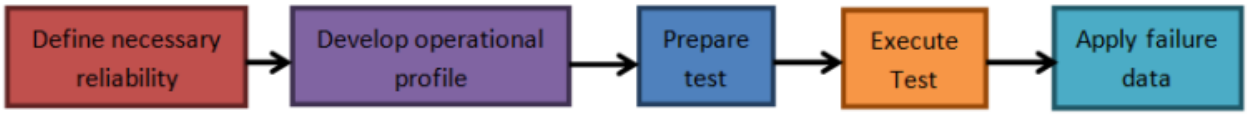


Figure 21: steps involved in reliability estimation tool

SOFTWARE TOOLS:

5-1. CASRE (Computer-Aided Software Reliability Estimation)

CASRE is implemented as a software reliability modeling tool that addresses the ease-of-use issue and other issues.

5-2. SMERFS (Statistical Modeling and Estimation of Reliability Functions For Software)

The software reliability prediction tool is SMERFS ,a well-known and widely accepted software application for evaluation of test data for failure rate and defect discovery rate prediction.

The version of SMERFS used in this study included a total of 15 different reliability growth models. The input to SMERFS is a set of values consisting

either of the time between discoveries of defects or the number of defects discovered per time period. SMERFS then uses maximum likelihood methods or least squares methods to estimate the parameters used for one or more of these models (depending on the type of input and user selected options).

Its output includes the parameter estimates, predicted values and a measure of the goodness-of-fit .

5-3. SoftRel

The software reliability process simulator SoftRel captures the effects of interrelationships among activities, and characterizes all events as piecewise-Poisson Markov processes with explicitly defined event rate functions. SoftRel simulates two types of failure events, namely, defects in specification documents and faults in code ,all considered to be in the same seriousness category, as reflected by the single set of "model" parameters The documentation' simulated by SoftRel consists only of requirements, design, interface specifications, and other entities whose absence or defective nature can root faults into subsequently produced code

Requirements analysis and design activities are currently combined in the document construction and integration phases in SoftRel. All defects occur either in proportion to the amount of new and reused documentation, to the amount that was changed, deleted, and added, or to the number of defects that were reworked.

The Characteristics of SoftRel:

- Console-based application written in C (about 1300 lines of code)
- Source code is available
- One input project file (formatted text)
- Generates one output file (CSV)

5-4. SRMP (Statistical Modeling and Reliability Program)

The SRMP was developed by the Reliability and Statistical Consultants, Limited of UK in 1988. SRMP is a command-line-oriented tool developed for an IBM PC/AT and also UNIX based workstations. SRMP contains nine models.

SRMP uses the maximum likelihood estimation technique to compute the model parameters, and provides the following reliability indicators:

- Reliability function
- Failure rate
- Mean time to failure
- Median time to failure, and
- The model parameters for each model.

SRMP requires an ASCII data file as an input. The file contains the name (or other identification of the project, the number of failures involved in the reliability analysis, and the inter failure times of all the failures. The input file also specifies the initial sample size used by SRMP for the initial fitting of each reliability model to the data. The remaining failures are used by SRMP for accessing a reliability model's prediction accuracy.

5-5. MEADEP (Measure and Dependability)

MEADEP is a failure data based dependability analysis and modeling tool. Dependability measures generated by MEADEP are either directly obtained from data, such as failure rate and event distribution, or evaluated by combined use of failure data and dependability models, such as system level reliability and availability. Thus, two basic types of input to MEADEP are:

- **Data**: structured failure reports containing information on failure time, location, type, impact and other failure characteristics
- **Models**: graphical specifications of dependability models including serial and parallel reliability blocks, weighted blocks, k-out-of-n blocks, and Markov reward chains.

The output of MEADEP consists of results obtained from data and results evaluated from models.

5-6. SOREL (Software Reliability Analysis and Prediction)

SoRel is a tool for Software Reliability analysis and prediction. It is composed of two parts allowing reliability growth tests and application of reliability growth models. It allows two kinds of failure data processing (inter-failure data and number of failures per unit of time, i.e. failure intensity data).

The results are available into two forms: Immediately on the screen (numerical results and curves), in the form of files which can be read and used by other Macintosh applications (Excel, Word…), numerical results and curves can thus be used directly for publications and reports. The program is modular and new reliability growth tests and models can easily be added.

5-7. SREPT (Software Reliability Estimation And Prediction Tool)

There is an increasing need for a tool that can be used to track the quality of a software product during the software development process, right from the architectural phase all the way up to the operational phase of the software. SREPT offers several techniques that can be used at various stages in the software life-cycle. Thus it makes it possible to monitor the quality of the entire software development process under a unified framework for software reliability estimation and prediction. SREPT combines the capabilities of the existing tools in a unified framework.

In addition, it offers the following features:

1. Provides a means of incorporating test coverage into finite failure NHPP (Non-Homogeneous Poisson Process) reliability growth models, thus improving the quality of estimation.
2. It offers a prediction system to take into account finite fault removal times as opposed to the conventional software reliability growth models which assume instantaneous and perfect fault removal. The user can specify the fault removal rate which will reflect the scheduling and resource allocation decisions.
3. It incorporates techniques for architecture-based reliability and performance prediction.

6- Failure Mode and Effect Analysis (FMEA)

Failure Mode and Effect Analysis (FMEA) is an engineering technique used to define, identify, and eliminate known and/or potential failures, problems, errors, and so on from the system, design, process, and/or service before they reach the customer.

FMEA is one of the most important early preventive actions in system, design, process, or service which will prevent failures and errors from occurring and reaching the customer

The FMEA will identify corrective actions required to prevent failures from reaching the customer, thereby assuring the highest durability, quality, and reliability possible in a product or service.

6-1 **A Good FMEA**:

1. Identifies known and potential failure modes.
2. Identifies the causes and effects of each failure mode.
3. Prioritizes the identified failure modes according to the Risk Priority Number (RPN)—the product of frequency of occurrence, severity, and detection.
4. Provides for problem follow-up and corrective action.

6-2 The Four Types of FMEAs are:

1. **System FMEA** (concept FMEA)—Used to analyze systems and subsystems in the early concept and design stage. A system FMEA focuses on potential failure modes between the functions of the system caused by system deficiencies. It includes the interactions between systems and elements of the system.

   The output of the system (concept) FMEA is:

   a. A potential list of failure modes ranked by the RPN.
   b. A potential list of system functions that could detect potential failure modes.
   c. A potential list of design actions to eliminate failure modes, safety issues, and reduce the occurrence.

   Benefits of the system FMEA are:

   a. Helps select the optimum system design alternative.
   b. Helps in determining redundancy.
   c. Helps in defining the basis for system level diagnostic procedures.

d. Increases the likelihood that potential problems will be considered.
e. Identifies potential system failures and their interaction with other systems or subsystems.

2. **Design FMEA**—Used to analyze products before they are released to manufacturing. A design FMEA focuses on failure modes caused by design deficiencies.

The output of the design FMEA is:

a. A potential list of failure modes ranked by the RPN.
b. A potential list of critical and/or significant characteristics.
c. A potential list of design actions to eliminate failure modes, safety issues, and reduce the occurrence.
d. A potential list of parameters for appropriate testing, inspection, and/or detection methods.
e. A potential list of recommended actions for the critical and significant characteristics.

The benefits of the design FMEA are that it:

a. Establishes a priority for design improvement actions.
b. Documents the rationale for changes.
c. Provides information to help through product design verification and testing.
d. Helps identify the critical or significant characteristics.
e. Assists in the evaluation of design requirements and alternatives.
f. Helps identify and eliminate potential safety concerns
g. Helps identify product failure early in the product development phase.

3. **Process FMEA**—Used to analyze manufacturing and assembly processes. A process FMEA focuses on failure modes caused by process or assembly deficiencies.

The output of the process FMEA is:

a. A potential list of failure modes ranked by the RPN.
b. A potential list of critical and/or significant characteristics.
c. A potential list of recommended actions to address the critical and significant characteristics.

The benefits of the process FMEA are that it:

a. Identifies process deficiencies and offers a corrective action plan.
b. Identifies the critical and/or significant characteristics and helps in developing control plans.
c. Establishes a priority of corrective actions.
d. Assists in the analysis of the manufacturing or assembly process.
e. Documents the rationale for changes.

4. Service FMEA—Used to analyze services before they reach the customer. A service FMEA focuses on failure modes (tasks, errors, mistakes) caused by system or process deficiencies.

The output of the service FMEA is:

a. A potential list of errors ranked by the RPN.
b. A potential list of critical or significant tasks, or processes.
c. A potential list of bottleneck processes or tasks.
d. A potential list to eliminate the errors.
e. A potential list of monitoring system/process functions.

The benefits of the service FMEA are that it:

a. Assists in the analysis of job flow.
b. Assists in the analysis of the system and/or process.
c. Identifies task deficiencies.
d. Identifies critical or significant tasks and helps in the development of control plans.
e. Establishes a priority for improvement actions.
f. Documents the rationale for changes.

6-3 The Process of Conducting an FMEA

To conduct an FMEA effectively, a systematic approach must be followed. The recommended approach is an seven-step method that facilitates the system, design, product, process, and service FMEA.

Step1- Select the team: Make sure the appropriate individuals are going to participate. The team must be cross-functional and multi-disciplined and the team members must be willing to contribute .

Step2- Functional block diagram and/or process flowchart: For system and design FMEAs the functional block diagram is applicable because the functional block diagram focuses the discussion on the system and design. For the process and service FMEAs the process flowchart is applicable, due to the fact that the process flowchart focuses the discussion on the process and service.

Step3- Prioritize: After the team understands the problem, the actual analysis begins. Frequent questions are: What part is important? , Where should the team begin?. Sometimes, this step is completely bypassed because the prioritization is de facto. The customer has identified the priority, or due to warranty cost or some other input the determination has been made by the management to start at a given point.

Step4- Data Collection: This is where the team begins to collect the data of the failures and categorizes them appropriately. At this point the team begins to fill in the FMEA form. The failures identified are the failure modes of the FMEA.

Step5- Analysis: Now the data are utilized for a resolution. Remember, the reason for the data is to gain information that is used to gain knowledge. Ultimately, that knowledge contributes to the decision. This flow can be shown as follows:

Data → Information → Knowledge → Decision → >>> Flow >>>> →

Step6- Results: The theme here is data driven. Based on the analysis, results are derived. The information from this step will be used to quantify the severity, occurrence, detection, and RPN. The appropriate columns of the FMEA will be completed.

Step7- Confirm/ Evaluate/ Measure—After the results have been recorded, it is time to confirm, evaluate, and measure the success or failure. This evaluation takes the form of three basic questions:

1. Is the situation better than before?
2. Is the situation worse than before?
3. Is the situation the same as before?

7- Software Reliability Models

Software reliability growth models can be used as an indication of the number of failures that may be encountered after the software has shipped and thus as an indication of whether the software is ready to ship. These models use system test data to predict the number of defects remaining in the software.

The utility of a software reliability growth model is related to its stability and predictive ability. Stability means that the model parameters should not significantly change as new data is added. Predictive ability means that the number of remaining defects predicted by the model should be close to the number found in field use.

Software reliability models are classified into four different types according to the phase of software development as follows:

a. Testing and debugging phase:

Faults are repaired without introducing new ones, increasing in that way the reliability of the system (reliability growth models).

b. Validation phase:

Faults are not corrected and may lead to the rejection of the software. This types of models describe systems for critical applications that must be highly reliable.

c. Operational phase:

System inputs are selected using a certain probability distribution for a certain (random) period of time.

d. Maintenance phase:

During this phase activities like fault correction or improvement of implemented features may take place and may modify the reliability of the system. The updated reliability can be estimated using the models for the validation phase.

Software Reliability Growth Models and Data

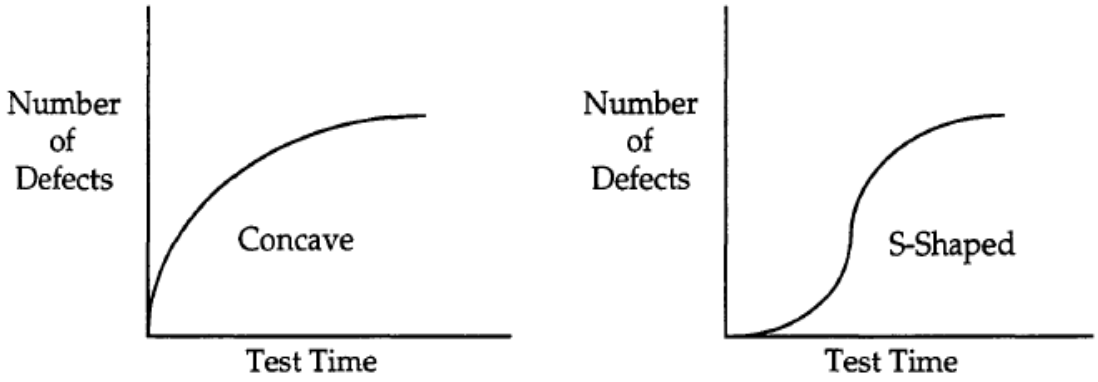
There are two relevant types of data for software reliability growth models. The first is the time at which the defect was discovered, and the second is the number of defects discovered.

Some examples of Software Reliability Growth Models :

1- Reliability Growth Models:

There are many different software reliability growth models, and many different ways to represent the data that is used to create those models.

Software reliability growth models have been grouped into two classes of models concave and S-shaped. The most important thing about both models is that they have the same asymptotic behavior, i.e., the defect detection rate decreases as the number of defects detected (and repaired) increases, and the total number of defects detected asymptotically approaches a finite value.



Concave and S-shaped models

This representation shows the expected number of defects at time t and is denoted μ(t), where t can be calendar time, execution time, or number of tests executed .

2- Exponential Failure Class Models:

The exponential model is one of the better known models and is often the basis of many other software reliability growth models. This group consists of all finite failure models which have exponential failure time distribution. Binomial and Poisson are two categorization of exponential model. The Binomial and Poisson types are based on per fault constant hazard rate. Hazards rate function is defined as the function of the remaining number of faults and the failure function is exponential.

$H(Z) = f(RNF) + f((\exp(FF))$

Where:

H(Z)= Hazard rate.

RNF= remaining number of faults.

FF= Failure function.

There are four types of exponential model:

a- **J-M Model (JMM):** The failure time is proportional to the remaining faults and taken as an exponential distribution.

$$(MTBF)_t = 1/(N-(i-1))$$

Where:

N= Total number of faults.
i= Number of faults occurrences.
MTBF=Mean Time between failure.
t=Time between the occurrence of the $(i-1)^{st}$ and i^{th} fault occurrences.

b- Execution Time Model (ETM): The intensity function is directly proportional to the number of faults remaining in the program and fault correction is proportional to the number of failure occurrence rate.

$$\mu(t)=\beta_0(1-\exp(-\beta_1 t))$$

Where:

$\mu(t)$= mean value function at time t.
β_0= total number of faults.

c- Hyper Exponential Model (HEM): The idea behind this model is that the different parts of the software experience an exponential failure rate. However the rate varies through these parts to ponder different behaviors Different failure rate are placed in different sections.

$$\lambda(t) = N \Sigma p_i \beta_i (\exp(-\beta_i t))$$

Where:

$\lambda(t)$ = Failure Intensity Function.
t = number of failure.
N = finite number of failures.
p_i = particular i^{th} class.
β_i = total number of i^{th} faults.

d- **Goel-Okumoto Model (GOM):** G-O model takes the number of faults per unit time as independent random variables. In this model the number of faults occurred within the time and model estimates the failure time. Delivery of software within cost estimates is also decided by this model.

$$\mu(t) = E_E (1-e^{-bt})$$

Where:

$\mu(t)$ = Predicted number of defects at time t.

EE = Expected total number of defects in the code in infinite time (it is usually finite).

b = Roundness factor/shape factor = the rate at which the failure rate decreases.

t = Calendar time/ execution time/ number of test runs.

3- Weibull and Gamma Failure Class Models:

Models under this category follow per fault failure Gamma distribution instead of exponential distribution. Weibull Model (WM): The model is used for hardware reliability. The model incorporates both increasing/decreasing and failure rate due to high flexibility. This model is a finite failure model.

$$\mathbf{MTTF = \int (1-F(t)) = \int \exp(-\beta_t{}^{\alpha})}$$

Where,

MTTF = Mean Time to Failure.

α, β = Weibull distribution parameters.

t= time of failure.

4- Infinite Failure Time Models:

Software is not completely error free when mean value function of a particular model tends to infinity. Models come under the category of Infinite Failure Time Model.

a. <u>Duane's Model (DM):</u> Duane observed that an erected line has been generated by comparing testing time with failure rate. Based on the observations of hardware reliability the same behavior has been observed for software & used for estimating software reliability.

$$\mathbf{\mu(t)/T = (\alpha T^{\beta})T}$$

where

$\mu(t)$ = Mean value function at time t.

$\alpha T\beta$ = Expected Number of failures by time t.

T = Total testing

b. <u>Geometric Model (GM):</u> The model is based on the version of J-M. The time between failures is an exponentially distributed and mean time failure decreased geometrically.

$$\mathbf{E(X_i) = 1/Z(t_{i-1})}$$

Where

E(Xi)= Expected time between failure.

Z(ti-1) =Fault detection rate.

c. **Logarithmic Poisson (LM):** The model assumes that code has an infinite number of failures. The model follows NHPP. When failure occur distribution decreases exponentially [17]. The possible number of failures over the time is a logarithmic function therefore it is called Logarithmic Poisson.

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

Where

$\mu(t)$=Mean value function at time t.

θ =failure rate decay parameter.

$\lambda(t)$=Failure intensity function.

5- Bayesian Models (BM):

The models are used by several organizations. BM incorporates past and current data. Prediction is done on the bases of number of faults that have been found & the amount of failure free operations.

L-V Reliability Growth Model: The model tries to account for fault generation in the fault correction process by allowing for the probability that the software program could become less reliable than before. For every correction of fault, a separate program has to write. Fault correction is obtained by fixing fault .

$$D(i) = \mu (1/ \lambda(i))$$

Where,

$\lambda(i)$ =Sequence of independent variable.

D (i) =Distribution for the i^{th} failure.