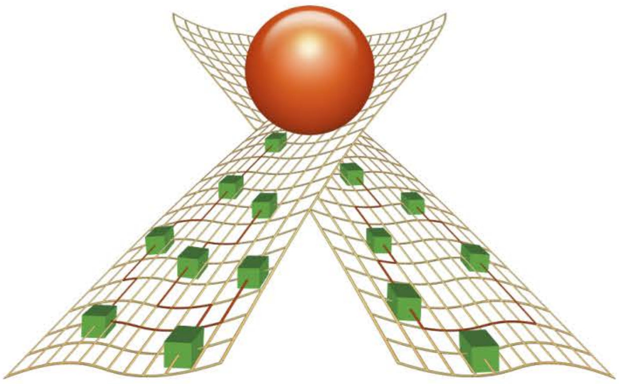


ReliaSoft

TOOLS TO EMPOWER
THE RELIABILITY PROFESSIONAL

System Analysis Reference

Reliability, Availability & Optimization

System Analysis Reference

ReliaSoft Corporation
Worldwide Headquarters
1450 South Eastside Loop
Tucson, Arizona 85710-6703, USA
http://www.ReliaSoft.com

Quick License Summary Overview	
You are Free to:	
	Share: Copy and redistribute the material in any medium or format
	Adapt: Remix, transform, and build upon the material
Under the following terms:	
	Attribution: You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. See example at http://www.reliawiki.org/index.php/Attribution_Example
	NonCommercial: You may not use this material for commercial purposes (sell or distribute for profit). Commercial use is the distribution of the material for profit (selling a book based on the material) or adaptations of this material.
	ShareAlike: If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

i. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.
j. **Licensor** means **ReliaSoft Corporation, 1450 Eastside Loop, Tucson, AZ 85710.**
k. **NonCommercial** means not primarily intended for or directed towards commercial advantage or monetary compensation. For purposes of this Public License, the exchange of the Licensed Material for other material subject to Copyright and Similar Rights by digital file-sharing or similar means is NonCommercial provided there is no payment of monetary compensation in connection with the exchange.
l. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.
m. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.
n. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. **License grant**.
    1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:
        A. reproduce and Share the Licensed Material, in whole or in part, for NonCommercial purposes only; and
        B. produce, reproduce, and Share Adapted Material for NonCommercial purposes only.
    2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.
    3. Term. The term of this Public License is specified in Section 6(a).
    4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.
    5. Downstream recipients.
        A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.
        B. Additional offer from the Licensor – Adapted Material. Every recipient of Adapted Material from You automatically receives an offer from the Licensor to exercise the Licensed Rights in the Adapted Material under the conditions of the Adapter's License You apply.

Contents

Chapter 1

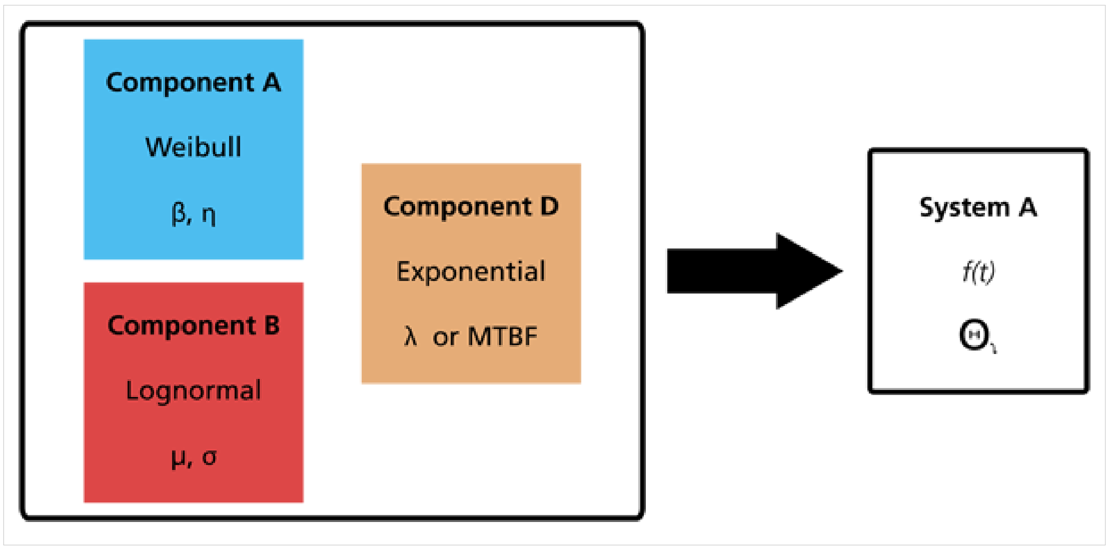
Basics of System Reliability Analysis

Overview

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.

For any life data analysis, the analyst chooses a point at which no more detailed information about the object of analysis is known or needs to be considered. At that point, the analyst treats the object of analysis as a "black box." The selection of this level (e.g., component, subassembly, assembly or system) determines the detail of the subsequent analysis.

In system reliability analysis, one constructs a "System" model from these component models. In other words in system reliability analysis we are concerned with the construction of a model (life distribution) 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 the figure below.



To accomplish this, the relationships between components are considered and decisions about the choice of components can be made to improve or optimize the overall system reliability, maintainability and/or availability. 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. Many other benefits of the system reliability analysis approach also exist and will be presented throughout this reference.

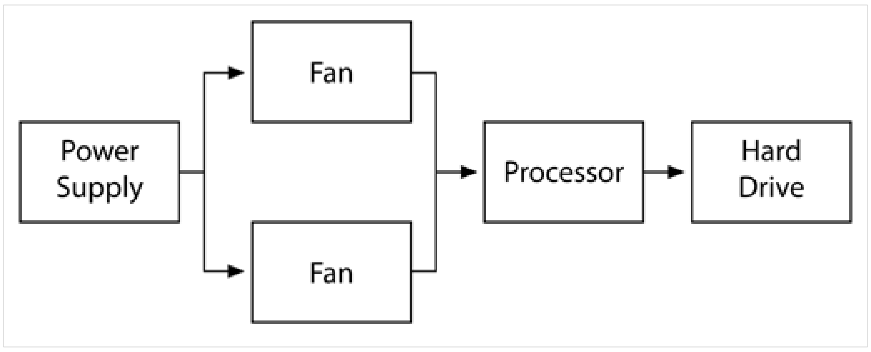
Systems

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. 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. The relationship between a system and its components is often misunderstood or oversimplified. For example, the following statement is not valid: All of the components in a system have a 90% reliability at a given time, thus the reliability of the system is 90% for that time. Unfortunately, poor understanding of the relationship between a system and its constituent components can result in statements like this being accepted as factual, when in reality they are false.

Reliability Block Diagrams (RBDs)

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 (connected). It should be noted that this may differ from how the components are physically connected. An RBD of a simplified computer system with a redundant fan configuration is shown below.

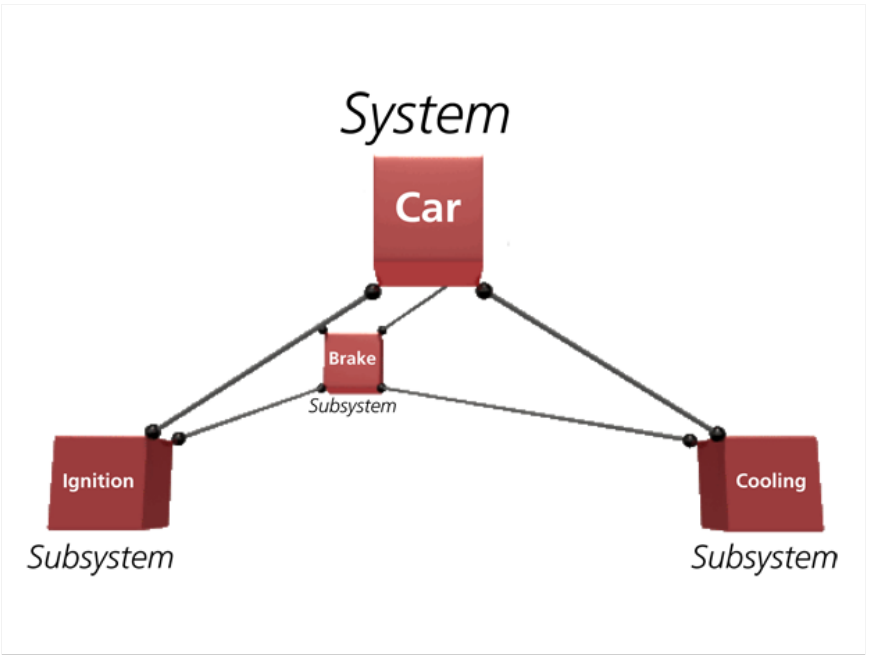


RBDs are constructed out of blocks. The blocks are connected with direction lines that represent the reliability relationship between the blocks.

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. The following figure shows two blocks, one representing a resistor and one representing a computer.

Resistor
Computer

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 the figure below. Each of these systems could have their 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 (e.g., fasteners), if so desired.



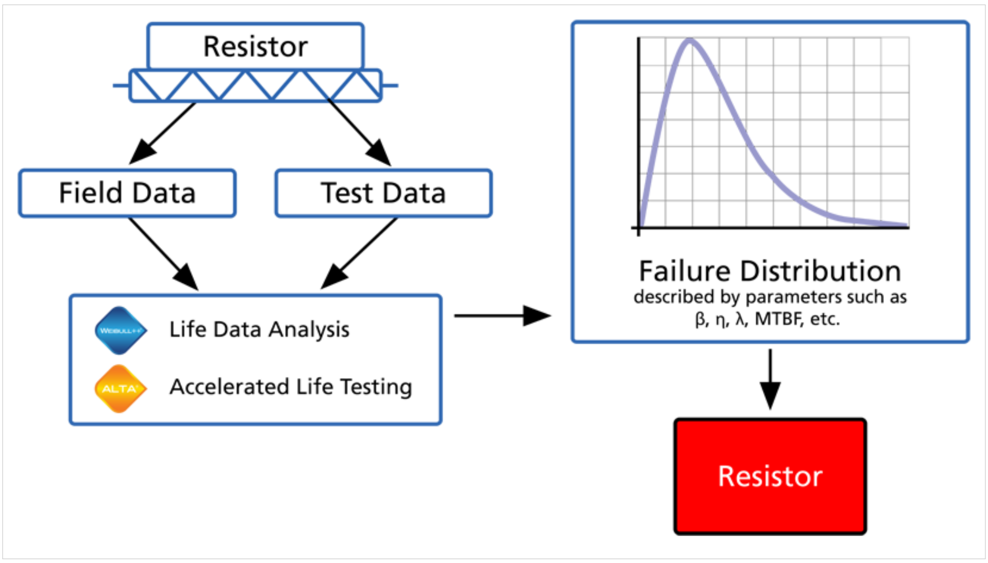
The level of granularity or detail that one chooses should be based on both the availability of data and on the lowest actionable item concept. To illustrate this concept, consider the aforementioned computer system shown earlier. When the computer manufacturer finds out that the hard drive is not as reliable as it should be and decides not to try to improve the reliability of the current hard drive but rather to get a new hard drive supplier, then the lowest actionable item is the hard drive. The hard drive supplier will then have actionable items inside the hard drive, and so forth.

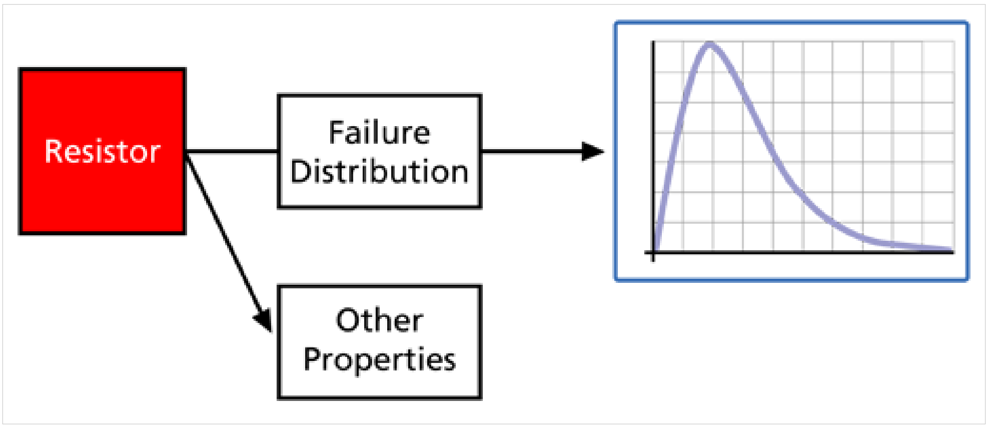
Block Failure Models

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 (i.e., 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

Additionally, component life data may also be provided by the manufacturer or supplier of the component/subsystem. Once the data set has been obtained, the life distribution of a component/subsystem can be estimated using ReliaSoft's Weibull++ or ALTA software. For example, consider a resistor that is part of a larger system to be analyzed. Failure data for this resistor can be obtained by performing in-house reliability tests and by observing the behavior of that type of resistor in the field. As shown below, a life distribution is then fitted to the data and the parameters are obtained. The parameters of that distribution represent the life distribution of that resistor block in the overall system RBD.





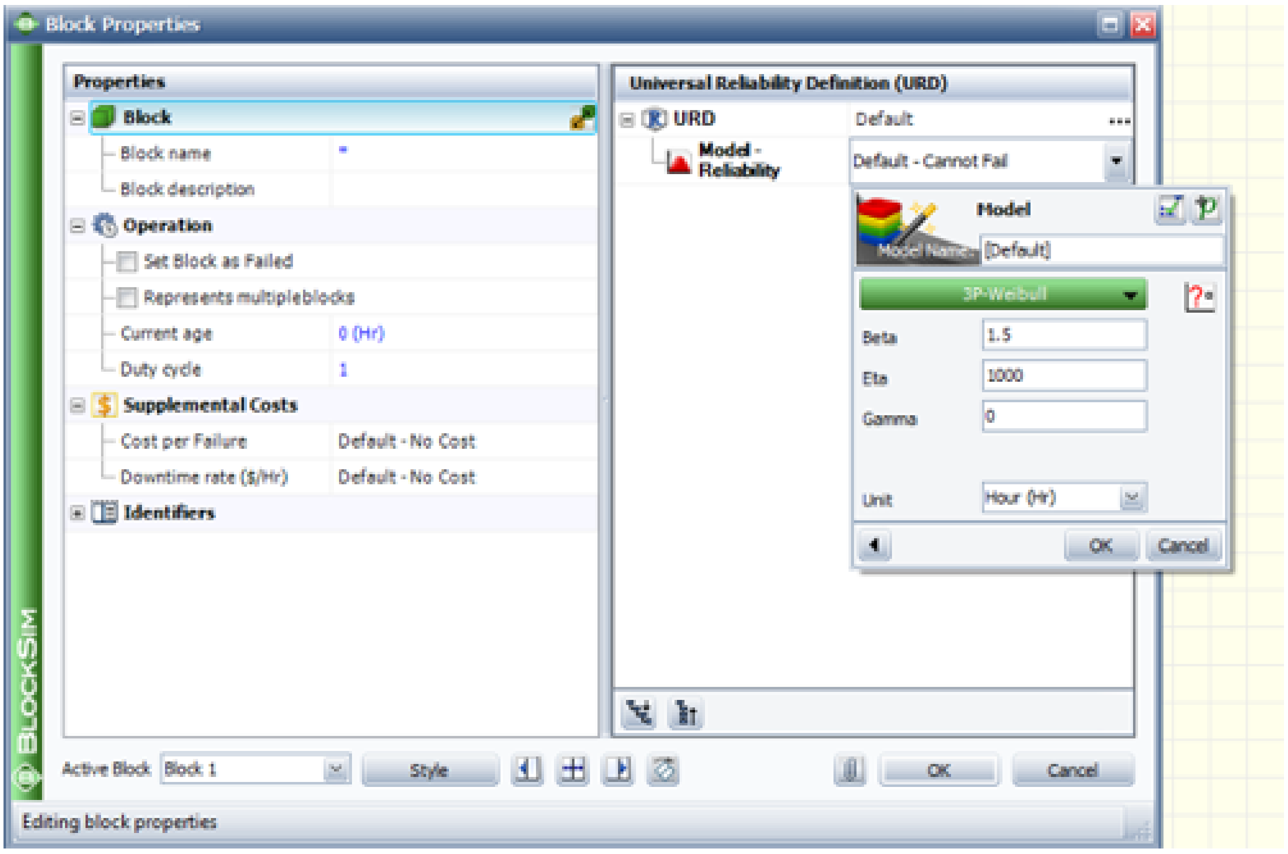
In the same manner, other types of information can also be obtained that can be used to define other block properties, such as the time-to-repair distribution (by analyzing the times-to-repair of each block instead of the times-to-failure), other maintenance requirements, throughput properties, etc. These block properties can then be used to perform a variety of analyses on the overall system to predict and/or optimize the system's reliability, maintainability, availability, spare parts utilization, throughput, etc.

Available Distributions

Because the failure properties of a component are best described by statistical distributions, the most commonly used life distributions are available in BlockSim. (For more information about these distributions, see Life Distributions.) The available distributions are:

1. 1 and 2 parameter exponential distributions
2. 1, 2 and 3 parameter Weibull distributions
3. Mixed Weibull distribution (with 2, 3 or 4 subpopulations)
4. Normal distribution
5. Lognormal distribution
6. Generalized-Gamma (i.e., G-Gamma) distribution
7. Gamma distribution
8. Logistic distribution
9. Loglogistic distribution
10. Gumbel distribution

The same distributions are also available as repair distributions and in other probabilistic property windows that we will discuss later. The first figure below illustrates the Block Properties window with the Weibull distribution assigned as the failure distribution while the second figure illustrates the Block Properties window with the normal distribution assigned as the repair distribution.



Block Properties

Properties	
Block	
Block name	*
Block description	
Operation	
Set Block as Failed	
Represents multiple blocks	
Current age	0 (Hr)
Duty cycle	1
Supplemental Costs	
Cost per Failure	Default - No Cost
Downtime rate ($/Hr)	Default - No Cost
Identifiers	

Universal Reliability Definition (URD)	
URD	Default
Model - Reliability	Default - Cannot Fail

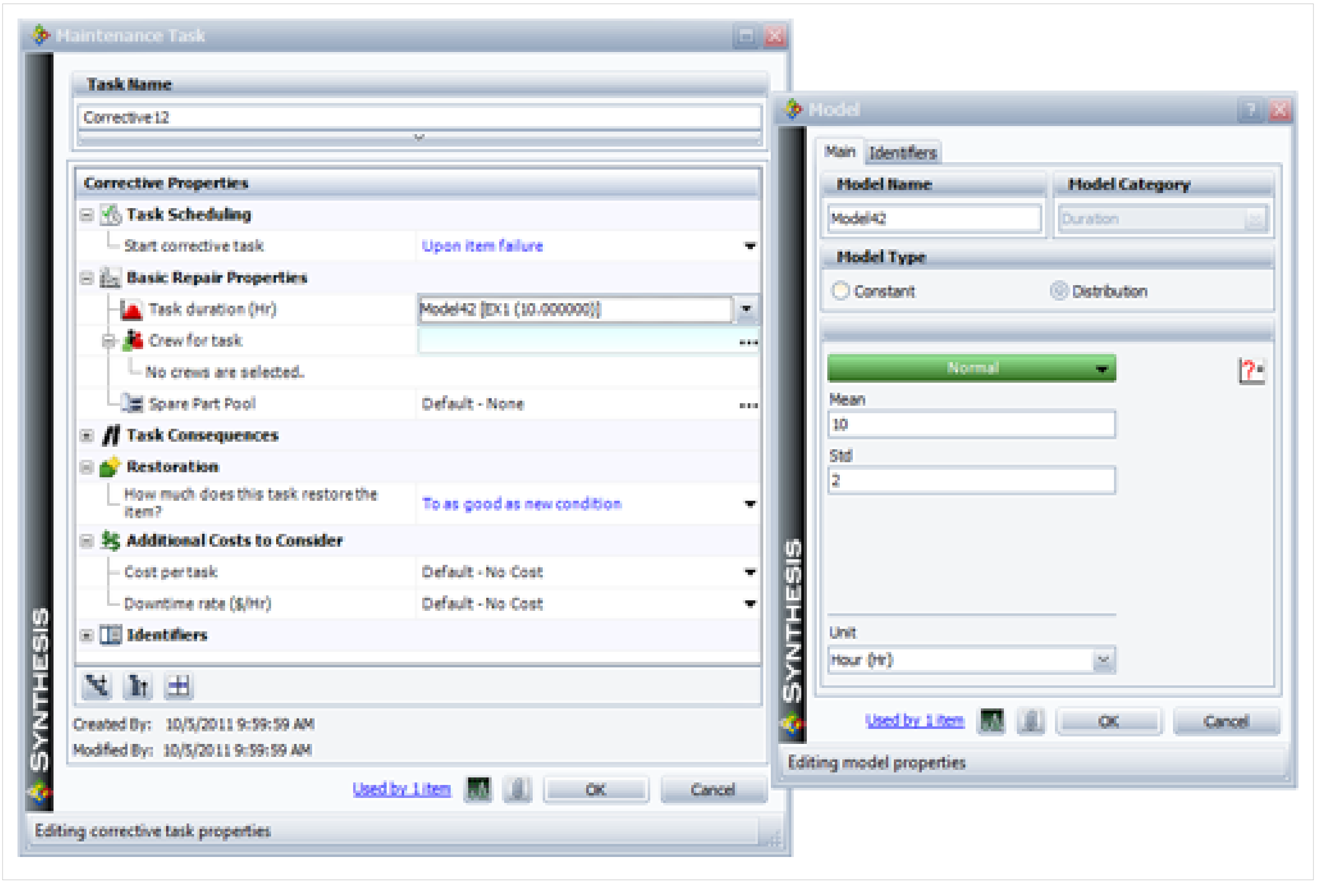
Model

Model Name: [Default]

3P-Weibull

Beta	1.5
Eta	1000
Gamma	0
Unit	Hour (Hr)

Editing block properties


Maintenance Task

Task Name

Corrective12

Corrective Properties

- Task Scheduling
  - Start corrective task: Upon item failure
- Basic Repair Properties
  - Task duration (Hr): Model42 [EX1 (10.000000)]
  - Crew for task
    - No crews are selected.
  - Spare Part Pool: Default - None
- Task Consequences
- Restoration
  - How much does this task restore the item?: To as good as new condition
- Additional Costs to Consider
  - Cost per task: Default - No Cost
  - Downtime rate ($/Hr): Default - No Cost
- Identifiers

Created By: 10/5/2011 9:59:59 AM

Modified By: 10/5/2011 9:59:59 AM

Editing corrective task properties

Model

Model Name: Model42

Model Category: Duration

Model Type: Constant [✓ Distribution]

Normal

Mean: 10

Std: 2

Unit: Hour (Hr)

Editing model properties

System Reliability Function

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 (i.e., 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.

The reliability-wise arrangement of components is directly related to the derived mathematical description of the system. The mathematical description of the system is the key to the determination of the reliability of the system. In fact, the system's reliability function is that mathematical description (obtained using probabilistic methods) and it defines the system reliability in terms of the component reliabilities. The result is an analytical expression that describes the reliability of the system as a function of time based on the reliability functions of its components. Statistical Background, RBDs and Analytical System Reliability and Time-Dependent System Reliability (Analytical) discuss this further. These chapters also offer derivations of needed equations and present examples.

Non-Repairable and Repairable Systems

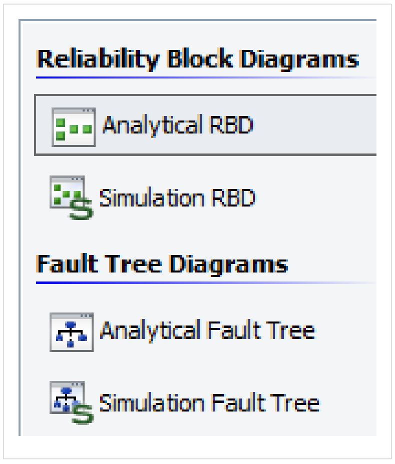
Systems can be generally classified into non-repairable and repairable systems. Non-repairable systems are those that do not get repaired when they fail. Specifically, the components of the system are not repaired or replaced when they fail. Most household products, for example, are non-repairable. This does not necessarily mean that they cannot be repaired, but rather that it does not make economic sense to do so. For example, repairing a four-year-old microwave oven is economically unreasonable, since the repair would cost approximately as much as purchasing a new unit.

On the other hand, repairable systems are those that get repaired when they fail. This is done by repairing or replacing the failed components in the system. An automobile is an example of a repairable system. If the automobile is rendered inoperative when a component or subsystem fails, that component is typically repaired or replaced rather than purchasing a new automobile. In repairable systems, two types of distributions are considered: failure distributions and repair distributions. A failure distribution describes the time it takes for a component to fail. A repair distribution describes the time it takes to repair a component (time-to-repair instead of time-to-failure). In the case of repairable systems, the failure distribution itself is not a sufficient measure of system performance, since it does not account for the repair distribution. A new performance criterion called availability can be calculated, which accounts for both the failure and repair distributions.

Repairable systems and availability will be discussed in Introduction to Repairable Systems and Repairable Systems Analysis Through Simulation.

BlockSim's Computation Modes

As shown in the figure below, BlockSim includes two independent computation modes: analytical and simulation. The analytical mode uses the exact reliability solutions for the system, employing the system's reliability function or cumulative density function (*cdf*). BlockSim can resolve even the most complex systems analytically and this method should be used when one is performing reliability analysis. In the context of BlockSim and this reference, we use the term *reliability analysis* to refer to all analyses that do not include repairs or restorations of the component. In contrast to the analytical mode, the simulation mode takes into account repair and restoration actions, including behaviors of crews, spare part pools, throughput, etc. Both of these methods will be explored in the chapters that follow.



When considering only the failure characteristics of the components, the analytical approach should be used. However, when both the failure and maintenance characteristics need to be considered, the simulation method must be used to take into account the additional events.

Analytical Calculations

In the analytical (or algebraic analysis) approach, the system's *pdf* is obtained analytically from each component's failure distribution using probability theory. In other words, the analytical approach involves the determination of a mathematical expression that describes the reliability of the system in terms the reliabilities of its components. Remember also that $cdf(t) = 1 - R(t)$.

The advantages of the analytical approach are:

- The mathematical expression for the system's *cdf* is obtained.
- Conditional reliability, warranty time and other calculations can be performed.
- Ancillary analyses can be performed, such as optimized reliability allocation, reliability importance computation of components, etc.

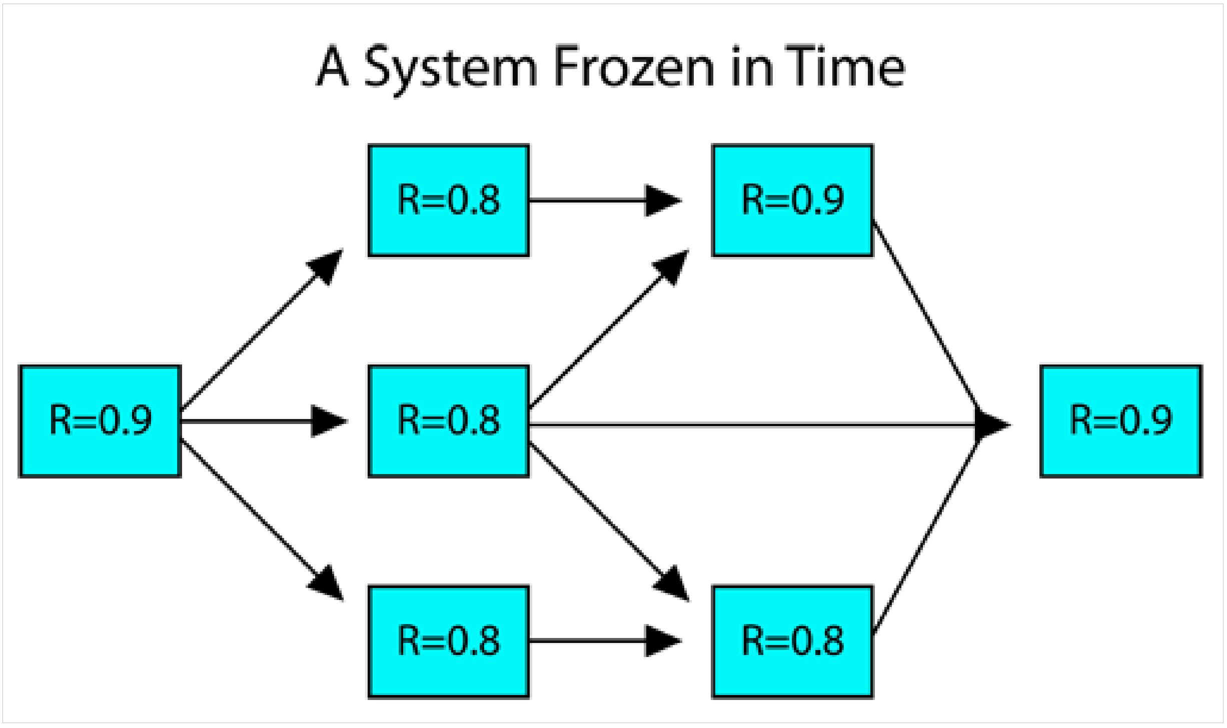
The disadvantage of the analytical approach is:

- Analyses that involve repairable systems with multiple additional events and/or other maintainability information are very difficult (if not impossible) to solve analytically. In these cases, analysis through simulation becomes necessary.

Two types of analytical calculations can be performed using RBDs (and BlockSim): static reliability calculations and time-dependent reliability calculations. Systems can contain static blocks, time-dependent blocks or a mixture of the two. Analytical computations are discussed in RBDs and Analytical System Reliability and Time-Dependent System Reliability (Analytical).

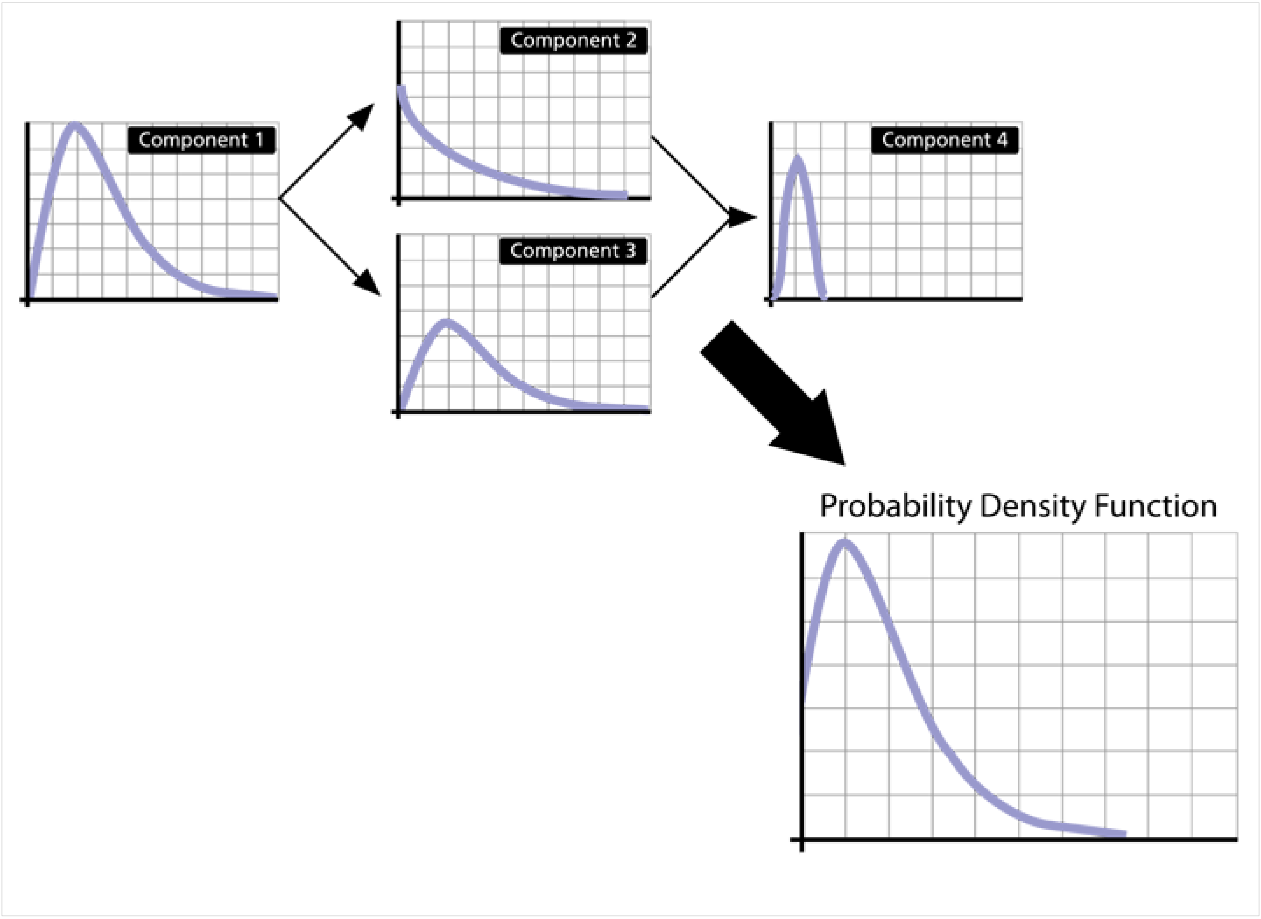
Static

Static analytical calculations are performed on RBDs that contain static blocks. A static block can be interpreted either as a block with a reliability value that is known only at a given time (but the block's entire distribution is unknown) or as a block with a probability of success that is constant with time. Static calculations can be performed both in the analytical mode and the simulation mode. The following figure illustrates a static RBD.



Time-Dependent

Time-dependent analysis looks at reliability as a function of time. That is, a known failure distribution is assigned to each component. The time scale in BlockSim can assume any quantifiable time measure, such as years, months, hours, minutes or seconds, and also units that are not directly related to time, such as cycles or miles of use. In many of the discussions and examples that follow, and to maintain generality, time units may be omitted or a general time unit (tu) may be utilized. It is very important to remember that even though any time unit may be used, the time units used throughout an analysis must be consistent in order to avoid incorrect results. The primary objective in system reliability analysis is to obtain a failure distribution of the entire system based on the failure distributions of its components, as illustrated below.



Simulation Calculations

If one includes information on the repair and maintenance characteristics of the components and resources available in the system, other information can also be analyzed/obtained, such as system availability, throughput, spare parts usage, life costs, etc. This can be accomplished through discrete event simulation.

In simulation, random failure times from each component's failure distribution are generated. These failure times are then combined in accordance with the way the components are reliability-wise arranged within the system. The overall results are analyzed in order to determine the behavior of the entire system.

The advantages of the simulation approach are:

- It can be used for highly complex scenarios involving a multitude of probabilistic events, such as corrective maintenance, preventive maintenance, inspections, imperfect repairs, crew response times, spare part availability, etc. When events such as these are considered, analytical solutions become impossible when dealing with real systems of sufficient complexity.

- The discrete event simulation also has the capabilities for:
  - Examining resource usage, efficiency and costs.
  - Optimizing procedures and resource allocation.
  - Analyzing relationships between systems and components.
  - Maximizing throughput.
  - Minimizing work downtimes.

The disadvantages of the simulation approach are:

- It can be time-consuming.
- The results are dependent on the number of simulations.
- There is a lack of repeatability in the results due to the random nature of data generation.

Simulation is discussed in the Repairable Systems Analysis Through Simulation and Throughput Analysis chapters.

Chapter 2

Statistical Background

This chapter presents a brief review of statistical principles and terminology. The objective of this chapter is to introduce concepts from probability theory and statistics that will be used in later chapters. As such, this chapter is not intended to cover this subject completely, but rather to provide an overview of applicable concepts as a foundation that you can refer to when more complex concepts are introduced.

If you are familiar with basic probability theory and life data analysis, you may wish to skip this chapter. If you would like additional information, we encourage you to review other references on the subject.

A Brief Introduction to Probability Theory

Basic Definitions

Before considering the methodology for estimating system reliability, some basic concepts from probability theory should be reviewed.

The terms that follow are important in creating and analyzing reliability block diagrams.

1. Experiment (E): An experiment is any well-defined action that may result in a number of outcomes. For example, the rolling of dice can be considered an experiment.
2. Outcome (O): An outcome is defined as any possible result of an experiment.
3. Sample space (S): The sample space is defined as the set of all possible outcomes of an experiment.
4. Event: An event is a collection of outcomes.
5. Union of two events $A$and $B\,(A\cup B)$: The union of two events $A$and $B$is the set of outcomes that belong to Aor Bor both.
6. Intersection of two events $A$and $B\,(A\cap B)$: The intersection of two events $A$and $B$is the set of outcomes that belong to both $A$and B.
7. Complement of event A ($\overline{A}$): A complement of an event Acontains all outcomes of the sample space, S, that do not belong to A.
8. Null event ($\varnothing$): A null event is an empty set that has no outcomes.
9. Probability: Probability is a numerical measure of the likelihood of an event relative to a set of alternative events. For example, there is a 50% probability of observing heads relative to observing tails when flipping a coin (assuming a fair or unbiased coin).

Example

Consider an experiment that consists of the rolling of a six-sided die. The numbers on each side of the die are the possible outcomes. Accordingly, the sample space is $S=\{1,2,3,4,5,6\}$.

Let Abe the event of rolling a 3, 4 or 6, $A=\{3,4,6\}$, and let Bbe the event of rolling a 2, 3 or 5, $B=\{2,3,5\}$.

1. The union of $A$and $B$is: $A\cup B=\{2,3,4,5,6\}$.
2. The intersection of $A$and $B$is: $A\cap B=\{3\}$.
3. The complement of $A$is: $\overline{A}=\{1,2,5\}$.

Probability Properties, Theorems and Axioms

The probability of an event A is expressed as $P(A)$ and has the following properties:

1. $0 \leq P(A) \leq 1$
2. $P(A) = 1 - P(\overline{A})$
3. $P(\varnothing) = 0$
4. $P(S) = 1$

In other words, when an event is certain to occur, it has a probability equal to 1; when it is impossible for the event to occur, it has a probability equal to 0.

It can also be shown that the probability of the union of two events A and B is:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Similarly, the probability of the union of three events, A, B and C is given by:

$$\begin{aligned} P(A \cup B \cup C) = & P(A) + P(B) + P(C) \\ & - P(A \cap B) - P(A \cap C) \\ & - P(B \cap C) + P(A \cap B \cap C) \end{aligned}$$

Mutually Exclusive Events

Two events A and B are said to be mutually exclusive if it is impossible for them to occur simultaneously ($A \cap B$ = $\varnothing$). In such cases, the expression for the union of these two events reduces to the following, since the probability of the intersection of these events is defined as zero.

$$P(A \cup B) = P(A) + P(B)$$

Conditional Probability

The conditional probability of two events A and B is defined as the probability of one of the events occurring, knowing that the other event has already occurred. The expression below denotes the probability of A occurring given that B has already occurred.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Note that knowing that event B has occurred reduces the sample space.

Independent Events

If knowing B gives no information about A, then the events are said to be *independent* and the conditional probability expression reduces to:

$$P(A|B) = P(A)$$

From the definition of conditional probability, $P(A|B) = \frac{P(A \cap B)}{P(B)}$ can be written as:

$$P(A \cap B) = P(A|B)P(B)$$

Since events A and B are independent, the expression reduces to:

$$P(A \cap B) = P(A)P(B)$$

If a group of n events A_i are independent, then:

$$P\left[\bigcap_{i=1}^{n} A_i\right] = \prod_{i=1}^{n} P(A_i)$$

As an illustration, consider the outcome of a six-sided die roll. The probability of rolling a 3 is one out of six or:

$$P(O=3)=1/6=0.16667$$

All subsequent rolls of the die are independent events, since knowing the outcome of the first die roll gives no information as to the outcome of subsequent die rolls (unless the die is loaded). Thus the probability of rolling a 3 on the second die roll is again:

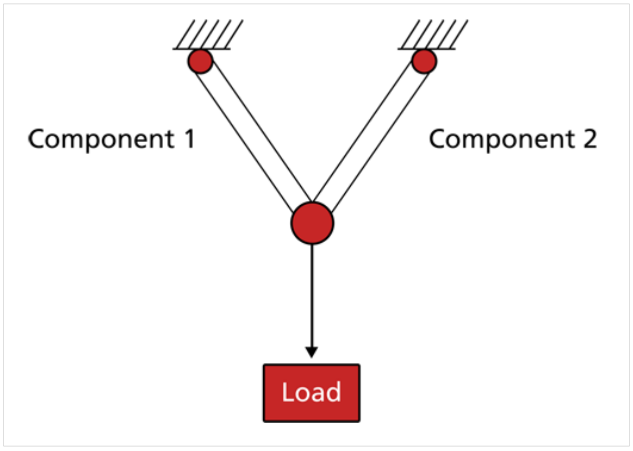
$$P(O=3)=1/6=0.16667$$

However, if one were to ask the probability of rolling a double 3 with two dice, the result would be:

$$\begin{aligned}0.16667\cdot 0.16667 &= 0.027778\\ &= \frac{1}{36}\end{aligned}$$

Example 1

Consider a system where two hinged members are holding a load in place, as shown next.



The system fails if either member fails and the load is moved from its position.

1. Let A =event of failure of Component 1 and let $\overline{A}$=the event of not failure of Component 1.
2. Let B =event of failure of Component 2 and let $\overline{B}$=the event of not failure of Component 2.

Failure occurs if Component 1 or Component 2 or both fail. The system probability of failure (or unreliability) is:

$$P_f=P(A\cup B)=P(A)+P(B)-P(A\cap B)$$

Assuming independence (or that the failure of either component is not influenced by the success or failure of the other component), the system probability of failure becomes the sum of the probabilities of $A$and Boccurring minus the product of the probabilities:

$$P_f=P(A\cup B)=P(A)+P(B)-P(A)P(B)$$

Another approach is to calculate the probability of the system not failing (i.e., the reliability of the system):

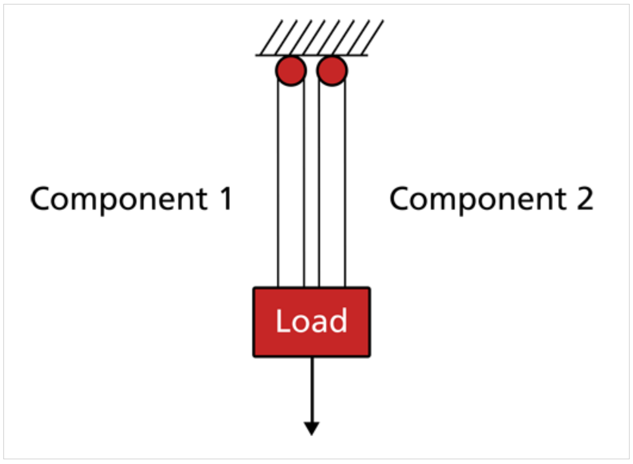
$$\begin{aligned}P(no\ failure) &= Reliability\\ &= P(\overline{A}\cap\overline{B})\\ &= P(\overline{A})P(\overline{B})\end{aligned}$$

Then the probability of system failure is simply 1 (or 100%) minus the reliability:

$$P_f=1-Reliability$$

Example 2

Consider a system with a load being held in place by two rigid members, as shown next.



- Let A =event of failure of Component 1.
- Let B =event of failure of Component 2.
- The system fails if Component 1 fails and Component 2 fails. In other words, both components must fail for the system to fail.

The system probability of failure is defined as the intersection of events $A$and B:

$$P_f = P(A \cap B))$$

Case 1

Assuming independence (i.e., either one of the members is sufficiently strong to hold the load in place), the probability of system failure becomes the product of the probabilities of $A$and Bfailing:

$$P_f = P(A \cap B) = P(A)P(B)$$

The reliability of the system now becomes:

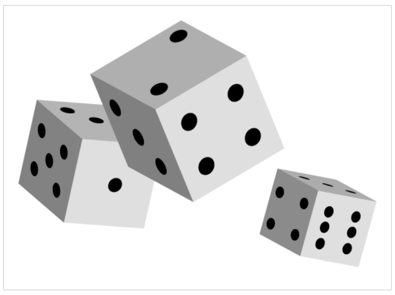
$$Reliability = 1 - P_f = 1 - P(A)P(B)$$

Case 2

If independence is not assumed (e.g., when one component fails the other one is then more likely to fail), then the simplification given in $Reliability = 1 - P_f = 1 - P(A)P(B)$is no longer applicable. In this case, $P_f = P(A \cap B))$must be used. We will examine this dependency in later sections under the subject of load sharing.

A Brief Introduction to Continuous Life Distributions

Random Variables



In general, most problems in reliability engineering deal with quantitative measures, such as the time-to-failure of a component, or qualitative measures, such as whether a component is defective or non-defective. We can then use a random variable $X$to denote these possible measures.

In the case of times-to-failure, our random variable $X$is the time-to-failure of the component and can take on an infinite number of possible values in a range from 0 to infinity (since we do not know the exact time *a priori*). Our component can be found failed at any time after time 0 (e.g., at 12 hours or at 100 hours and so forth), thus Xcan take on any value in this range. In this case, our random variable $X$is said to be a *continuous random variable*. In this reference, we will deal almost exclusively with continuous random variables.

In judging a component to be defective or non-defective, only two outcomes are possible. That is, $X$is a random variable that can take on one of only two values (let's say defective = 0 and non-defective = 1). In this case, the variable is said to be a discrete random variable.

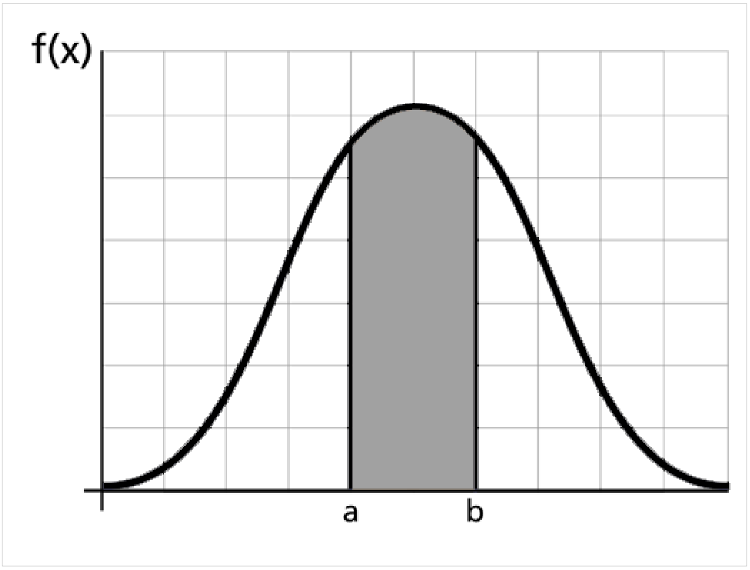
The Probability Density Function and the Cumulative Distribution Function

The probability density function (*pdf*) and cumulative distribution function (*cdf*) are two of the most important statistical functions in reliability and are very closely related. When these functions are known, almost any other reliability measure of interest can be derived or obtained. We will now take a closer look at these functions and how they relate to other reliability measures, such as the reliability function and failure rate.

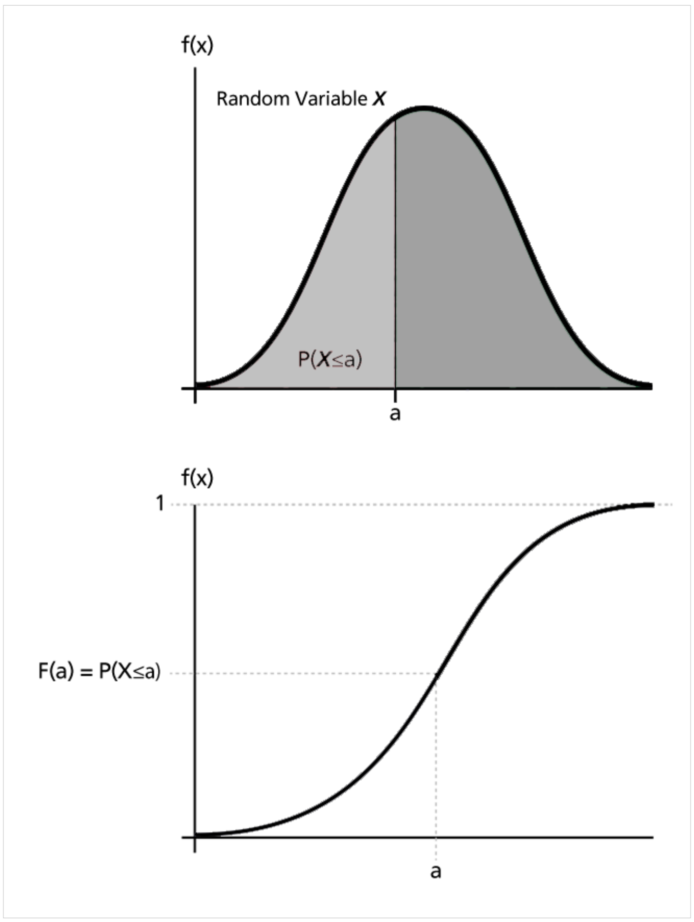
From probability and statistics, given a continuous random variable X,we denote:

- The probability density function, *pdf*, as $f(x)$.
- The cumulative distribution function, *cdf*, as $F(x)$.

The *pdf* and *cdf* give a complete description of the probability distribution of a random variable. The following figure illustrates a *pdf*.



The next figures illustrate the *pdf* - *cdf* relationship.



If X is a continuous random variable, then the *pdf* of X is a function, $f(x)$, such that for any two numbers, a and b with $a \leq b$:

$$P(a \leq X \leq b) = \int_a^b f(x)dx$$

That is, the probability that X takes on a value in the interval $[a, b]$ is the area under the density function from a to b, as shown above. The *pdf* represents the relative frequency of failure times as a function of time.

The *cdf* is a function, $F(x)$, of a random variable X, and is defined for a number x by:

$$F(x) = P(X \leq x) = \int_0^x f(s)ds$$

That is, for a number x, $F(x)$ is the probability that the observed value of X will be at most x. The *cdf* represents the cumulative values of the *pdf*. That is, the value of a point on the curve of the *cdf* represents the area under the curve to the left of that point on the *pdf*. In reliability, the *cdf* is used to measure the probability that the item in question will fail before the associated time value, t, and is also called *unreliability*.

Note that depending on the density function, denoted by $f(x)$, the limits will vary based on the region over which the distribution is defined. For example, for the life distributions considered in this reference, with the exception of the normal distribution, this range would be $[0, +\infty]$.

Mathematical Relationship: *pdf* and *cdf*

The mathematical relationship between the *pdf* and *cdf* is given by:

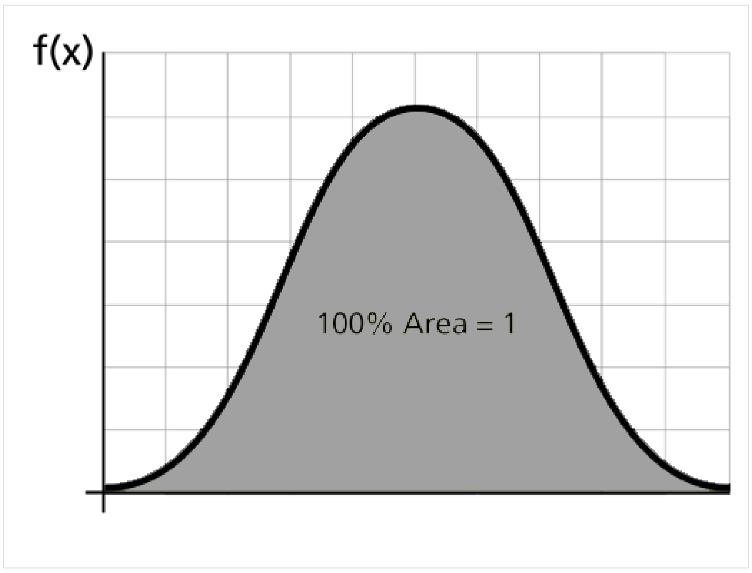
$$F(x) = \int_0^x f(s)ds$$

where s is a dummy integration variable.

Conversely:

$$f(x) = \frac{d(F(x))}{dx}$$

The *cdf* is the area under the probability density function up to a value of x. The total area under the *pdf* is always equal to 1, or mathematically:



$$\int_{-\infty}^{+\infty} f(x)dx = 1$$

The well-known normal (or Gaussian) distribution is an example of a probability density function. The *pdf* for this distribution is given by:

$$f(t) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}$$

where μ is the mean and σ is the standard deviation. The normal distribution has two parameters, μ and σ.

Another is the lognormal distribution, whose *pdf* is given by:

$$f(t) = \frac{1}{t \cdot \sigma'\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{t'-\mu'}{\sigma'}\right)^2}$$

where μ' is the mean of the natural logarithms of the times-to-failure and σ' is the standard deviation of the natural logarithms of the times-to-failure. Again, this is a 2-parameter distribution.

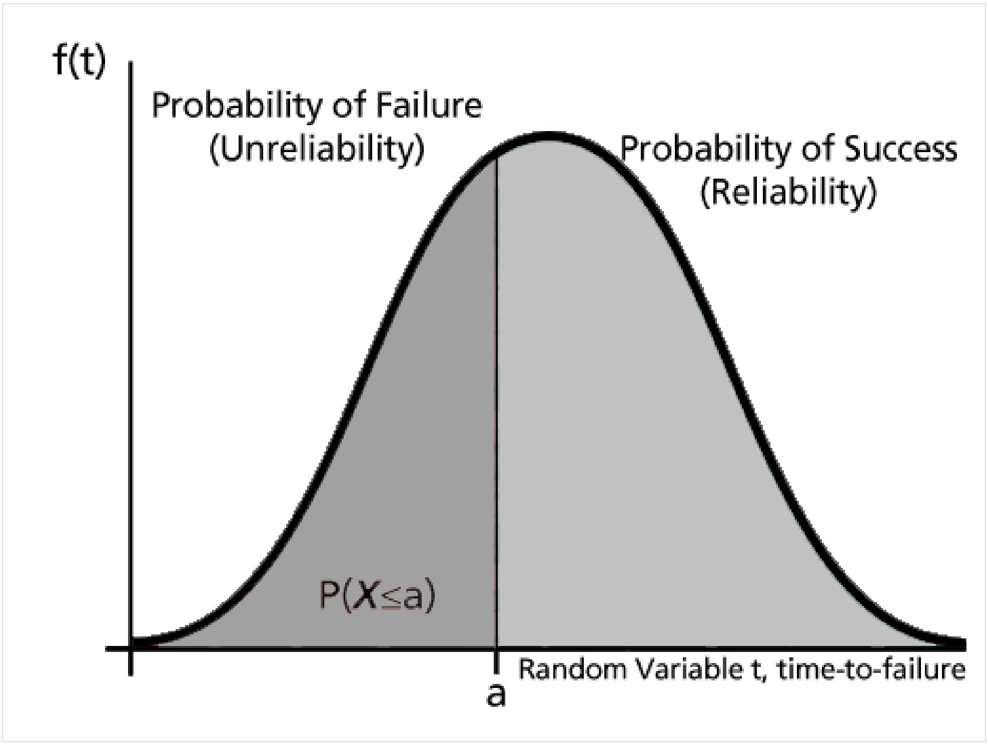
Reliability Function

The reliability function can be derived using the previous definition of the cumulative distribution function, $F(x)=\int_0^x f(s)ds$. From our definition of the *cdf*, the probability of an event occurring by time t is given by:

$$F(t)=\int_0^t f(s)ds$$

Or, one could equate this event to the probability of a unit failing by time t.

Since this function defines the probability of failure by a certain time, we could consider this the unreliability function. Subtracting this probability from 1 will give us the reliability function, one of the most important functions in life data analysis. The reliability function gives the probability of success of a unit undertaking a mission of a given time duration. The following figure illustrates this.


f(t)
Probability of Failure (Unreliability)
Probability of Success (Reliability)
P(X≤a)
a
Random Variable t, time-to-failure

To show this mathematically, we first define the unreliability function, $Q(t)$, which is the probability of failure, or the probability that our time-to-failure is in the region of 0 and t. This is the same as the *cdf*. So from $F(t)=\int_0^t f(s)ds$:

$$Q(t)=F(t)=\int_0^t f(s)ds$$

Reliability and unreliability are the only two events being considered and they are mutually exclusive; hence, the sum of these probabilities is equal to unity.

Then:

$$\begin{aligned} Q(t)+R(t) &= 1 \\ R(t) &= 1-Q(t) \\ R(t) &= 1-\int_0^t f(s)ds \\ R(t) &= \int_t^\infty f(s)ds \end{aligned}$$

Conversely:

$$f(t) = -\frac{d(R(t))}{dt}$$

Conditional Reliability Function

Conditional reliability is the probability of successfully completing another mission following the successful completion of a previous mission. The time of the previous mission and the time for the mission to be undertaken must be taken into account for conditional reliability calculations. The conditional reliability function is given by:

$$R(T,t) = \frac{R(T+t)}{R(T)}$$

Failure Rate Function

The failure rate function enables the determination of the number of failures occurring per unit time. Omitting the derivation, the failure rate is mathematically given as:

$$\lambda(t) = \frac{f(t)}{R(t)}$$

This gives the instantaneous failure rate, also known as the hazard function. It is useful in characterizing the failure behavior of a component, determining maintenance crew allocation, planning for spares provisioning, etc. Failure rate is denoted as failures per unit time.

Mean Life (MTTF)

The mean life function, which provides a measure of the average time of operation to failure, is given by:

$$\overline{T} = m = \int_0^{\infty} t \cdot f(t)dt$$

This is the expected or average time-to-failure and is denoted as the MTTF (Mean Time To Failure).

The MTTF, even though an index of reliability performance, does not give any information on the failure distribution of the component in question when dealing with most lifetime distributions. Because vastly different distributions can have identical means, it is unwise to use the MTTF as the sole measure of the reliability of a component.

Median Life

Median life, $\tilde{T}$, is the value of the random variable that has exactly one-half of the area under the *pdf* to its left and one-half to its right. It represents the centroid of the distribution. The median is obtained by solving the following equation for $\breve{T}$. (For individual data, the median is the midpoint value.)

$$\int_{-\infty}^{\breve{T}} f(t)dt = 0.5$$

Modal Life (or Mode)

The modal life (or mode), $\tilde{T}$, is the value of Tthat satisfies:

$$\frac{d\left[f(t)\right]}{dt} = 0$$

For a continuous distribution, the mode is that value of $t$that corresponds to the maximum probability density (the value at which the *pdf* has its maximum value, or the peak of the curve).

Lifetime Distributions

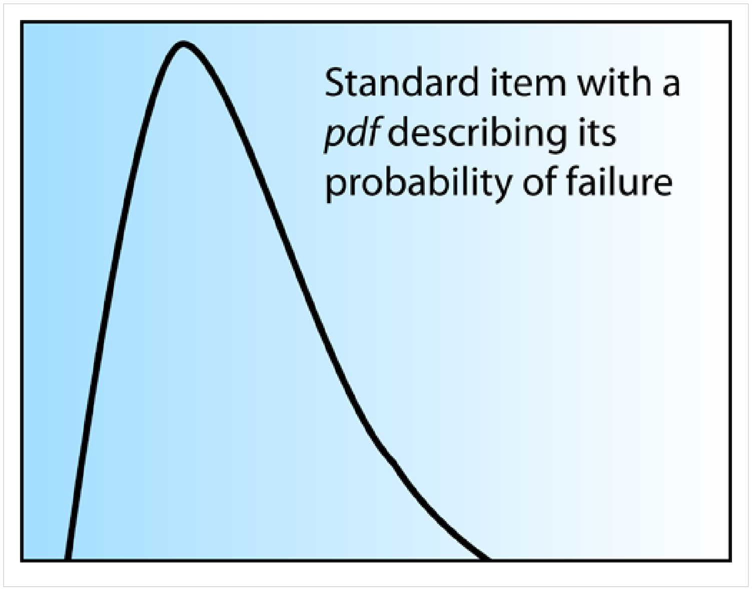
A statistical distribution is fully described by its *pdf*. In the previous sections, we used the definition of the *pdf* to show how all other functions most commonly used in reliability engineering and life data analysis can be derived. The reliability function, failure rate function, mean time function, and median life function can be determined directly from the *pdf* definition, or $f(t)$. Different distributions exist, such as the normal (Gaussian), exponential, Weibull, etc., and each has a predefined form of $f(t)$that can be found in many references. In fact, there are certain references that are devoted exclusively to different types of statistical distributions. These distributions were formulated by statisticians, mathematicians and engineers to mathematically model or represent certain behavior. For example, the Weibull distribution was formulated by Waloddi Weibull and thus it bears his name. Some distributions tend to better represent life data and are most commonly called "lifetime distributions".

A more detailed introduction to this topic is presented in Life Distributions.

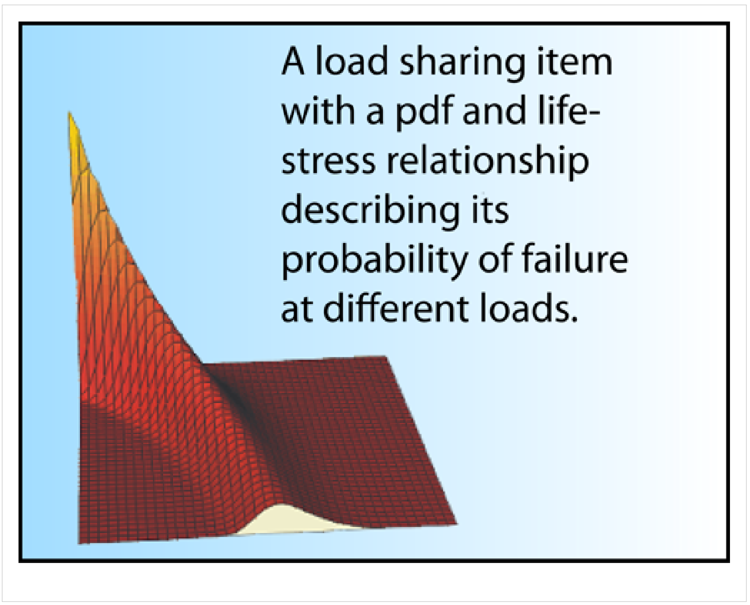
A Brief Introduction to Life-Stress Relationships

In certain cases when one or more of the characteristics of the distribution change based on an outside factor, one may be interested in formulating a model that includes both the life distribution and a model that describes how a characteristic of the distribution changes. In reliability, the most common "outside factor" is the stress applied to the component. In system analysis, stress comes into play when dealing with units in a load sharing configuration. When components of a system operate in a load sharing configuration, each component supports a portion of the total load for that aspect of the system. When one or more load sharing components fail, the operating components must take on an increased portion of the load in order to compensate for the failure(s). Therefore, the reliability of each component is dependent upon the performance of the other components in the load sharing configuration.

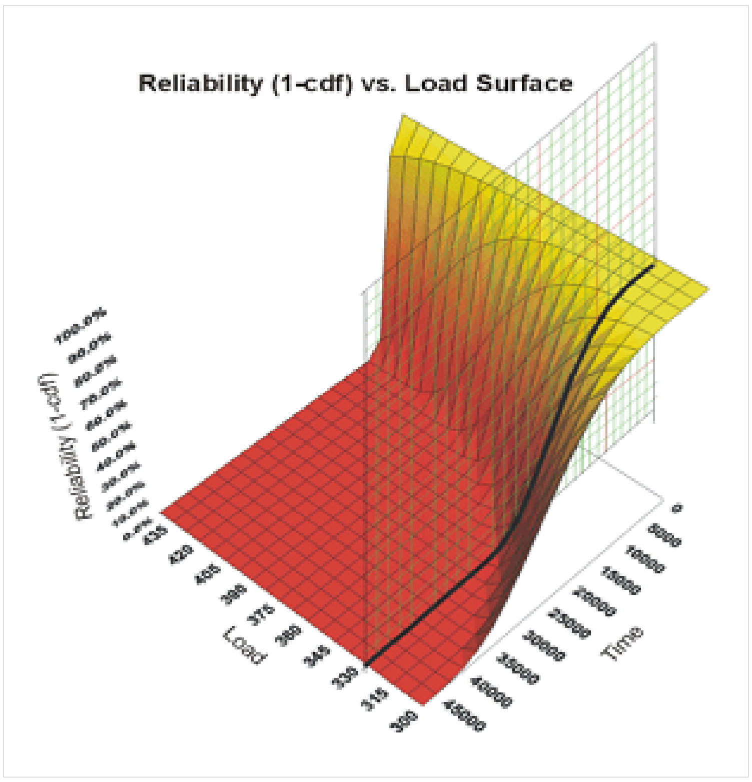
Traditionally in a reliability block diagram, one assumes independence and thus an item's failure characteristics can be fully described by its failure distribution. However, if the configuration includes load sharing redundancy, then a single failure distribution is no longer sufficient to describe an item's failure characteristics. Instead, the item will fail differently when operating under different loads and the load applied to the component will vary depending on the performance of the other component(s) in the configuration. Therefore, a more complex model is needed to fully describe the failure characteristics of such blocks. This model must describe both the effect of the load (or stress) on the life of the product and the probability of failure of the item at the specified load. The models, theory and methodology used in Quantitative Accelerated Life Testing (QALT) data analysis can be used to obtain the desired model for this situation. The objective of QALT analysis is to relate the applied stress to life (or a life distribution). Identically in the load sharing case, one again wants to relate the applied stress (or load) to life. The following figure graphically illustrates the probability density function (*pdf*) for a standard item, where only a single distribution is required.


Standard item with a *pdf* describing its probability of failure

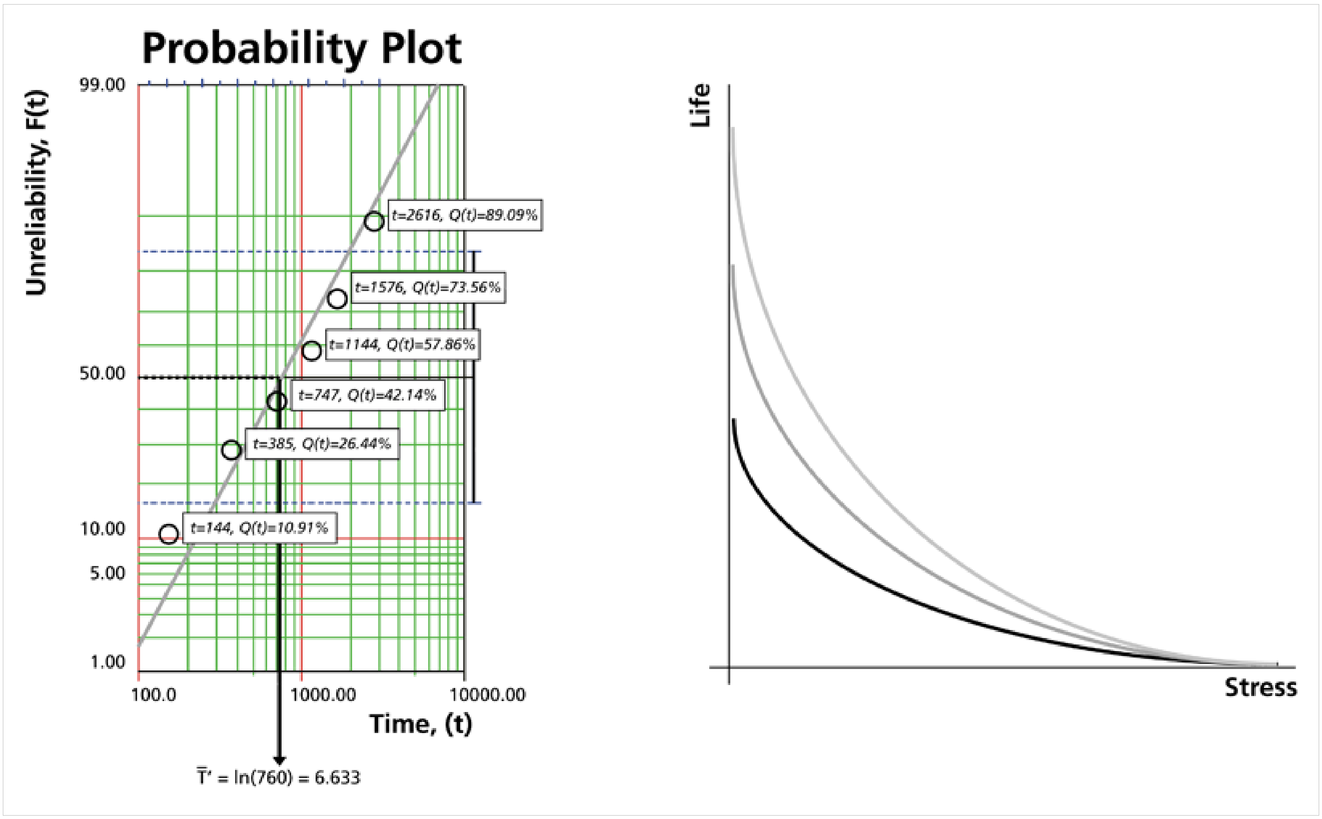
The next figure represents a load sharing item by using a 3-D surface that illustrates the *pdf* , load and time.


A load sharing item with a pdf and life-stress relationship describing its probability of failure at different loads.

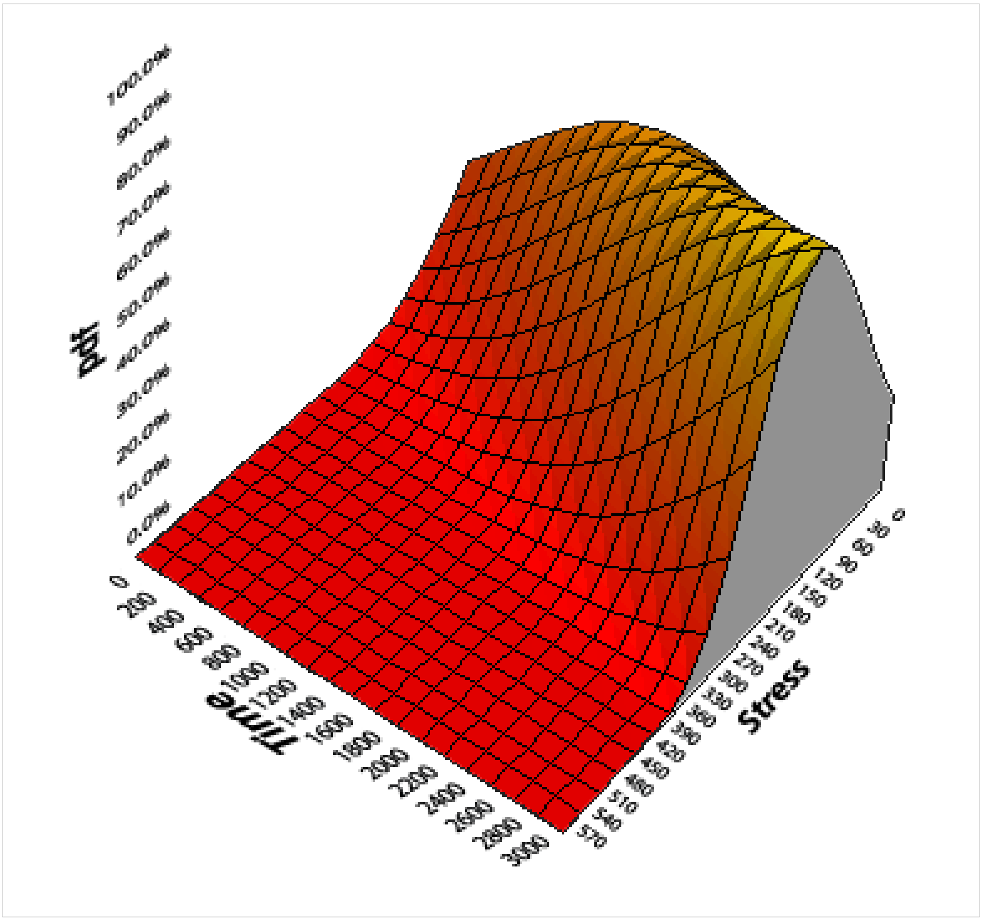
The following figure shows the reliability curve for a load sharing item vs. the applied load.



To formulate the model, a life distribution is combined with a life-stress relationship. The distribution choice is based on the product's failure characteristics while the life-stress relationship is based on how the stress affects the life characteristics. The following figure graphically shows these elements of the formulation.



The next figure shows the combination of both an underlying distribution and a life-stress model by plotting a *pdf* against both time and stress.



The assumed underlying life distribution can be any life distribution. The most commonly used life distributions include the Weibull, the exponential and the lognormal. The life-stress relationship describes how a specific life characteristic changes with the application of stress. The life characteristic can be any life measure such as the mean, median, $R(x)$, $F(x)$, etc. It is expressed as a function of stress. Depending on the assumed underlying life distribution, different life characteristics are considered. Typical life characteristics for some distributions are shown in the next table.

Distribution	***Parameters***	***Life Characteristic***
Weibull	$\beta *$, η	Scale parameter, η
Exponential	λ	Mean Life, ($1/\lambda$)
Lognormal	$\bar{T}$, $\sigma *$	Median, $\breve{T}$
*Usually assumed constant		

For example, when considering the Weibull distribution, the scale parameter, η, is chosen to be the life characteristic that is stress-dependent while $\beta$is assumed to remain constant across different stress levels. A life-stress relationship is then assigned to η.

For a detailed discussion of this topic, see ReliaSoft's Accelerated Life Testing Data Analysis Reference.

Chapter 3

RBDs and Analytical System Reliability

An overall system reliability prediction can be made by looking at the reliabilities of the components that make up the whole system or product. In this chapter, 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, also known as structural properties (Leemis [17]). Unless explicitly stated, the components will be assumed to be statistically independent.

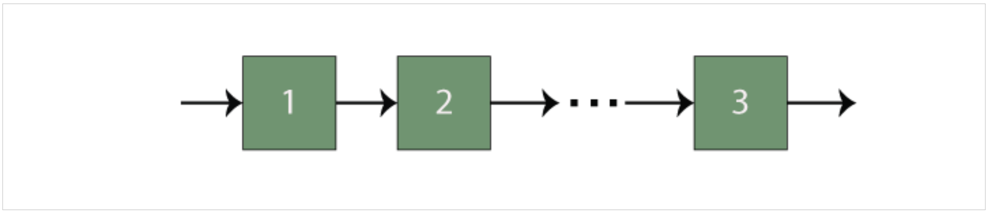
Component Configurations

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.

Each of these configurations will be presented, along with analysis methods, in the sections that follow.

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. In other words, all of the units in a series system must succeed for the system to succeed.

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

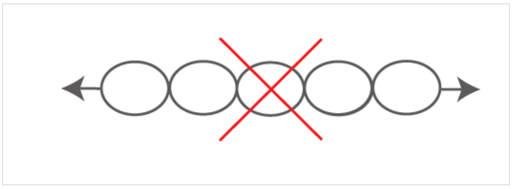
Three subsystems are reliability-wise in series and make up a system. Subsystem 1 has a reliability of 99.5%, subsystem 2 has a reliability of 98.7% and subsystem 3 has a reliability of 97.3% for a mission of 100 hours. What is the overall reliability of the system for a 100-hour mission?

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

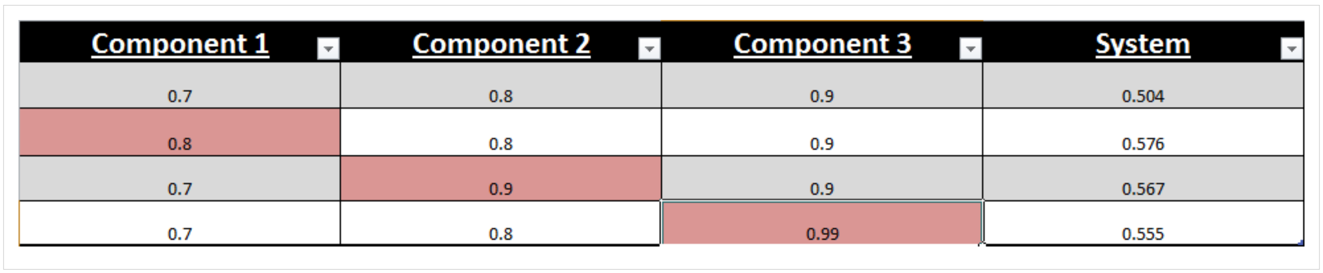
In a series configuration, the component with the least reliability has the biggest effect on the system's reliability. There is a saying that a chain is only as strong as its weakest link. This is a good example of the effect of a component in a series system. In a chain, all the rings are in series 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 a result, the reliability of a series system is always less than the reliability of the least reliable component.



Example: Effect of a Component's Reliability in a Series System

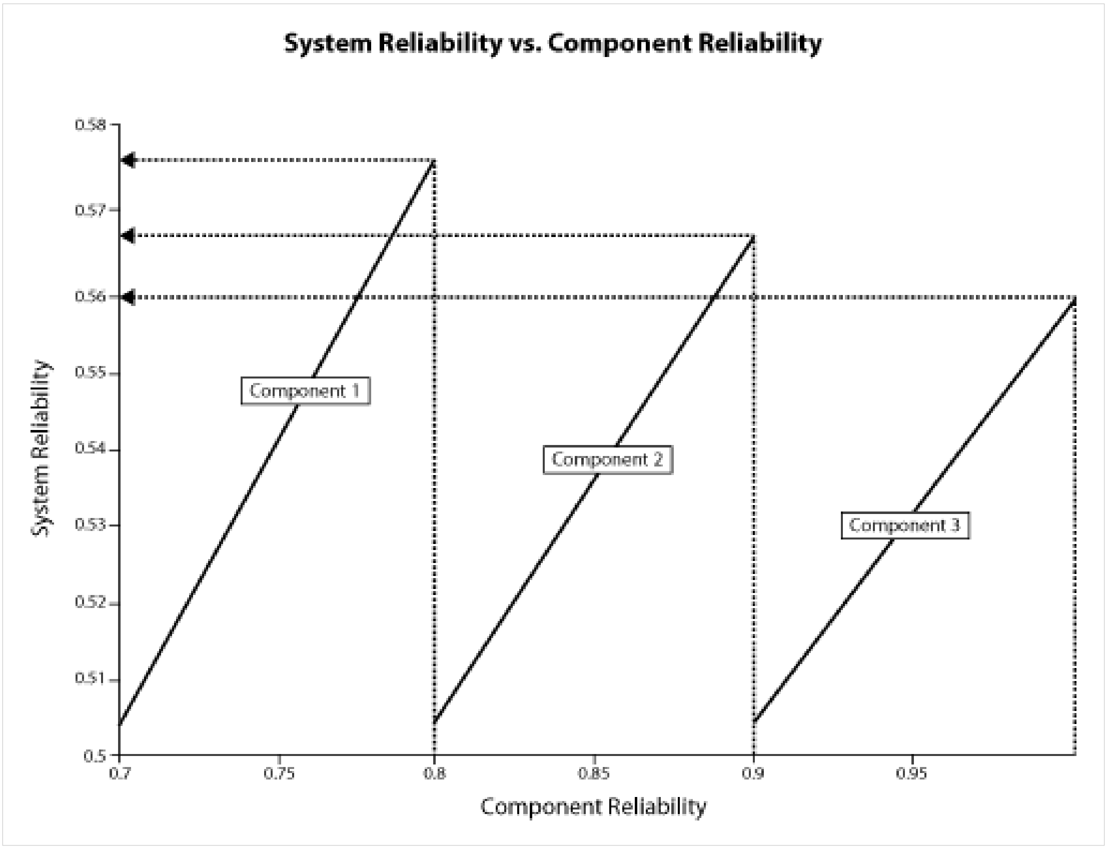
Consider three components arranged reliability-wise in series, where $R_1 = 70\%$, $R_2 = 80\%$and $R_3 = 90\%$ (for a given time). In the following table, we can examine the effect of each component's reliability on the overall system reliability. The first row of the table shows the given reliability 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 rows 3 and 4 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 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%.

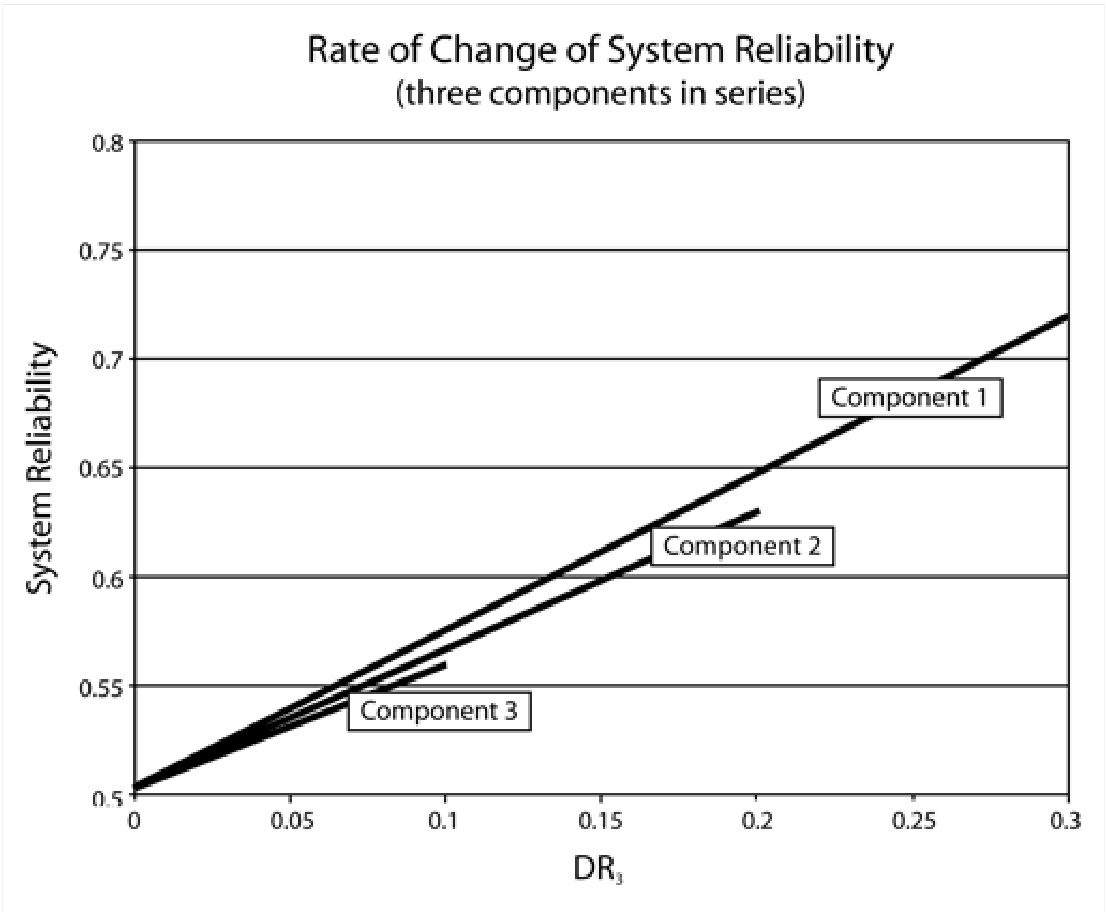
Table 1: System Reliability for Combinations of Component Reliabilities



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

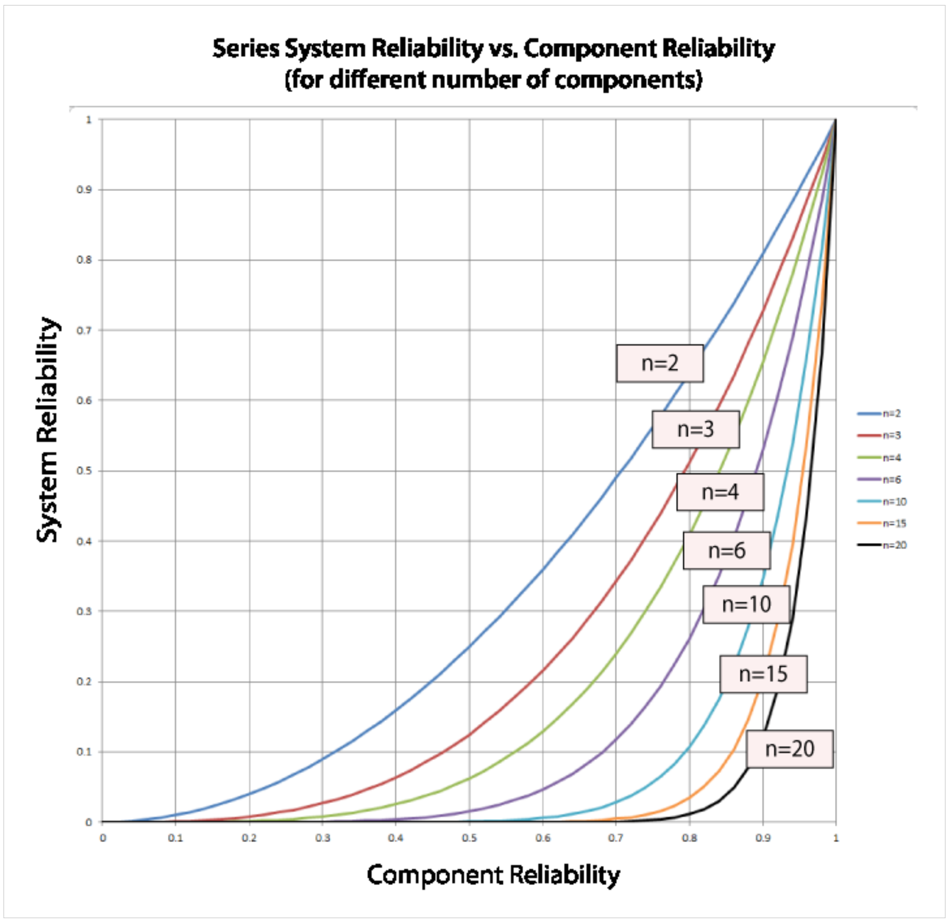
This conclusion can also be illustrated graphically, as shown in the following figure. Note the slight difference in the slopes of the three lines. The difference in these slopes represents the difference in the effect of each of the components on the overall system reliability. In other words, the system reliability's rate of change with respect to each component's change in reliability is different. This observation will be explored further when the importance measures of components are considered in later chapters. The rate of change of the system's reliability with respect to each of the components is also plotted. 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*.


System Reliability vs. Component Reliability
System Reliability
Component Reliability
Component 1
Component 2
Component 3
0.58
0.57
0.56
0.55
0.54
0.53
0.52
0.51
0.5
0.7
0.75
0.8
0.85
0.9
0.95


Rate of Change of System Reliability
(three components in series)
System Reliability
DR_3
Component 1
Component 2
Component 3
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0
0.05
0.1
0.15
0.2
0.25
0.3

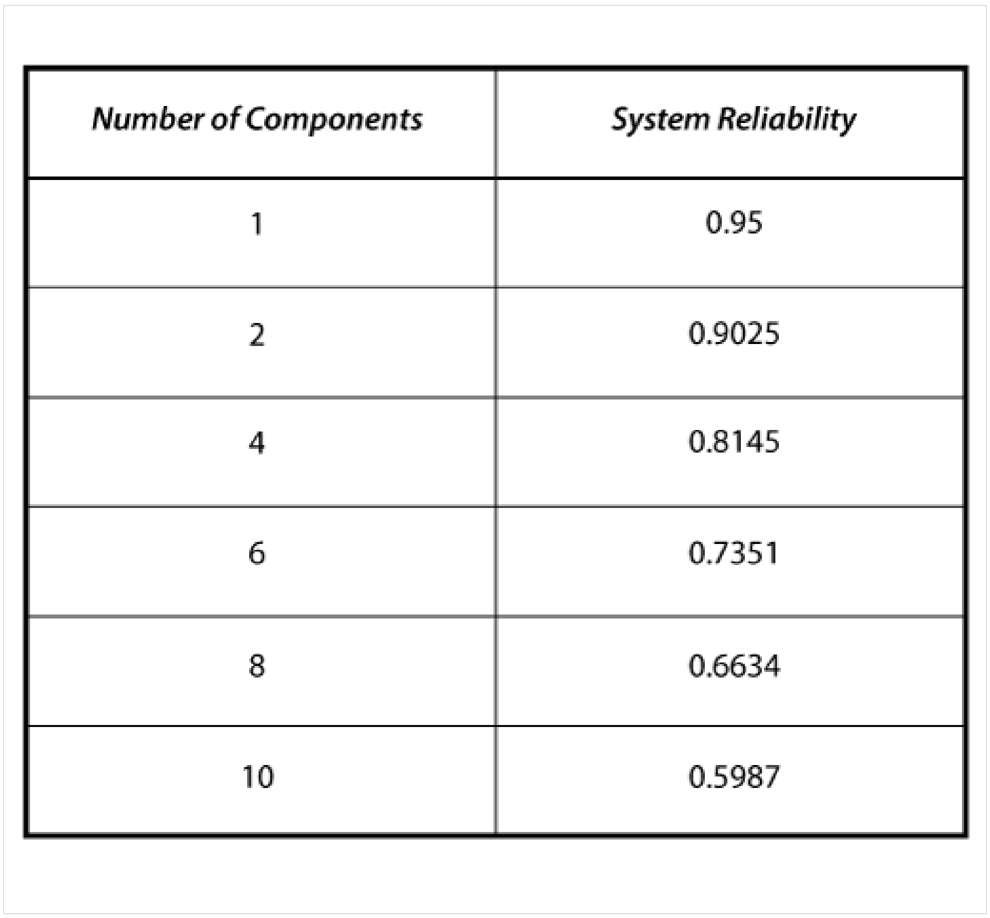
Effect of Number of Components in a Series System

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. The following figure illustrates the effect of the number of components arranged reliability-wise in series on the system's reliability for different component reliability values. 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.

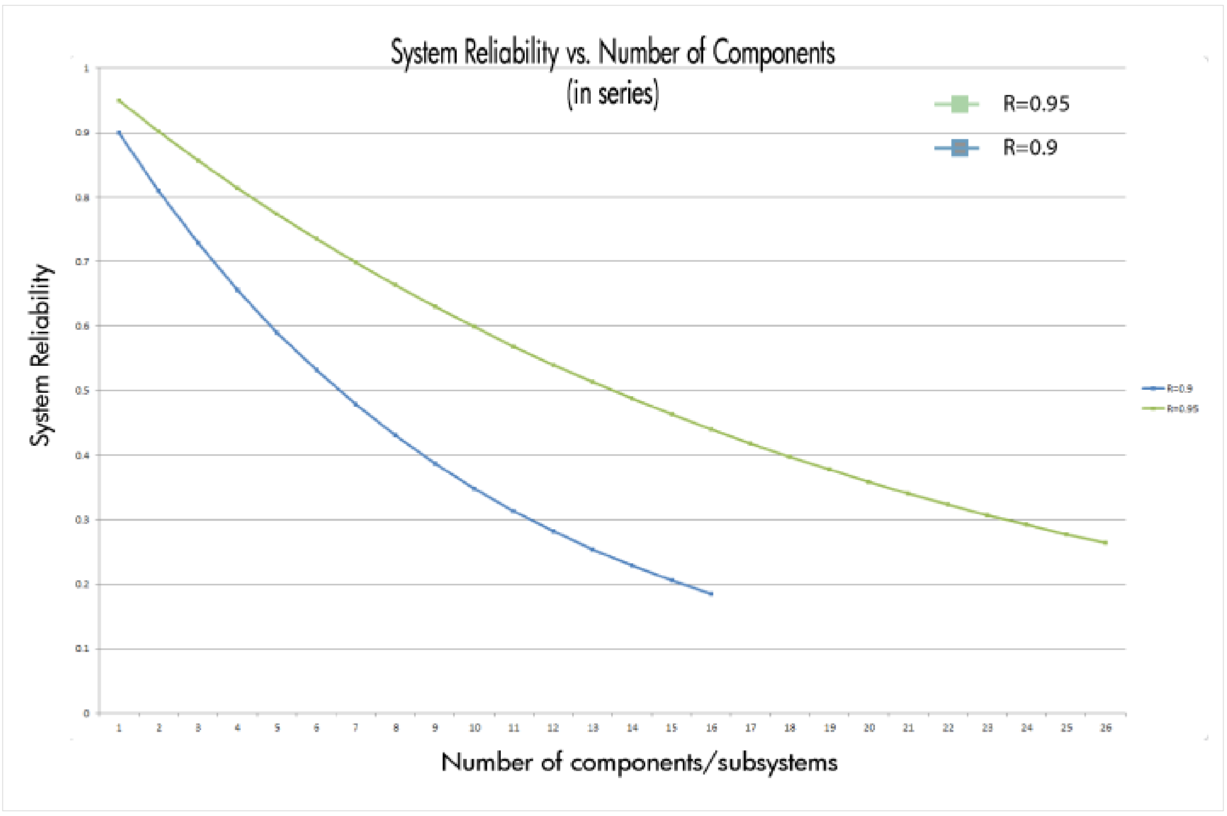


Example: Effect of the Number of Components in a Series System

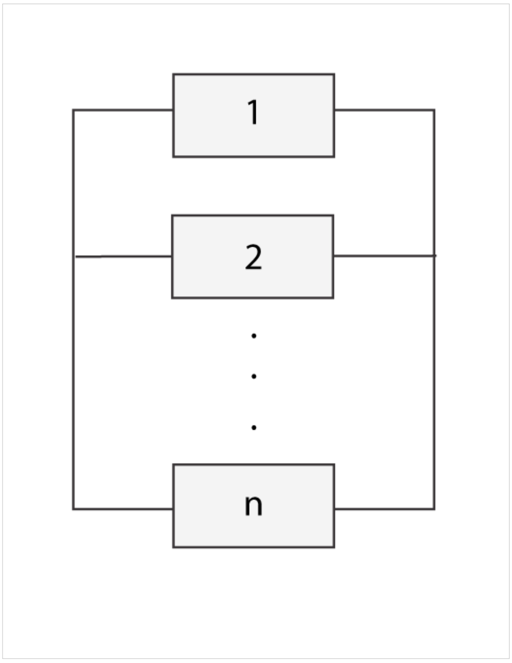
Consider a system that consists of a single component. The reliability of the component is 95%, thus the reliability of the system is 95%. What would the reliability of the system be if there were more than one component (with the same individual reliability) in series? The following table shows the effect on the system's reliability by adding consecutive components (with the same reliability) in series. The plot illustrates the same concept graphically for components with 90% and 95% reliability.



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



Simple Parallel Systems



In a simple parallel system, as shown in the figure on the right, 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. It is 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.

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. The unreliability of 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:

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

Observe the contrast with the series system, in which the system reliability was the product of the component reliabilities; whereas the parallel system has the overall system unreliability 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: Calculating the Reliability with Components in Parallel

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:

$$\begin{aligned} R_s &= 1-(1-0.9950)\cdot(1-0.9870)\cdot(1-0.9730) \\ &= 1-0.000001755 \\ &= 0.999998245 \end{aligned}$$

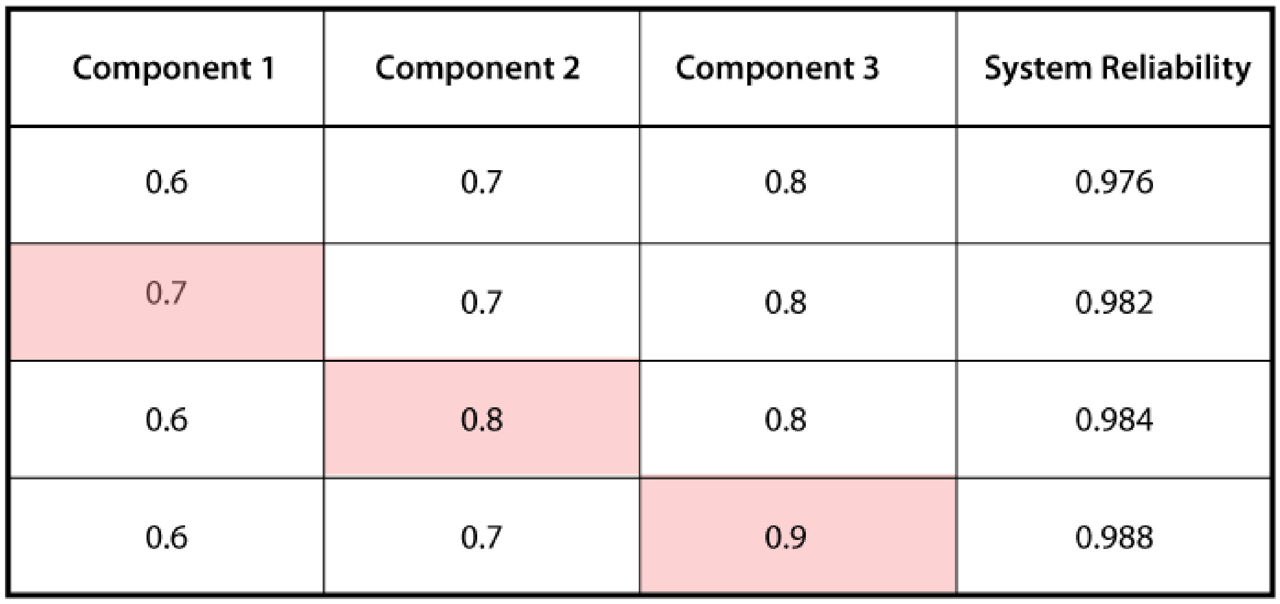
Effect of Component Reliability in a Parallel Configuration

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

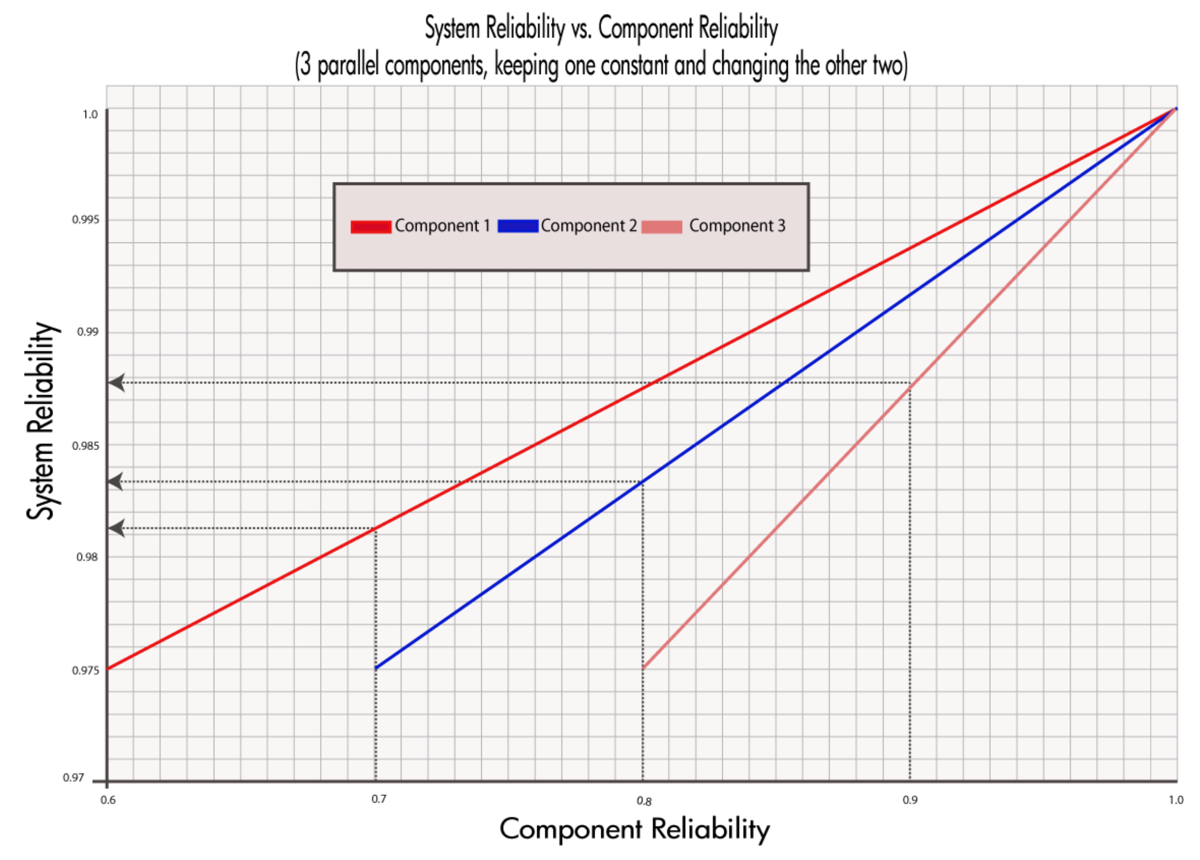
Consider three components arranged reliability-wise in parallel with $R_1 = 60\%$, $R_2 = 70\%$and $R_3 = 80\%$(for a given time). The corresponding reliability for the system is $R_s = 97.6\%$. In the following table, we can examine the effect of each component's reliability on the overall system reliability. 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 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



Component 1	Component 2	Component 3	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

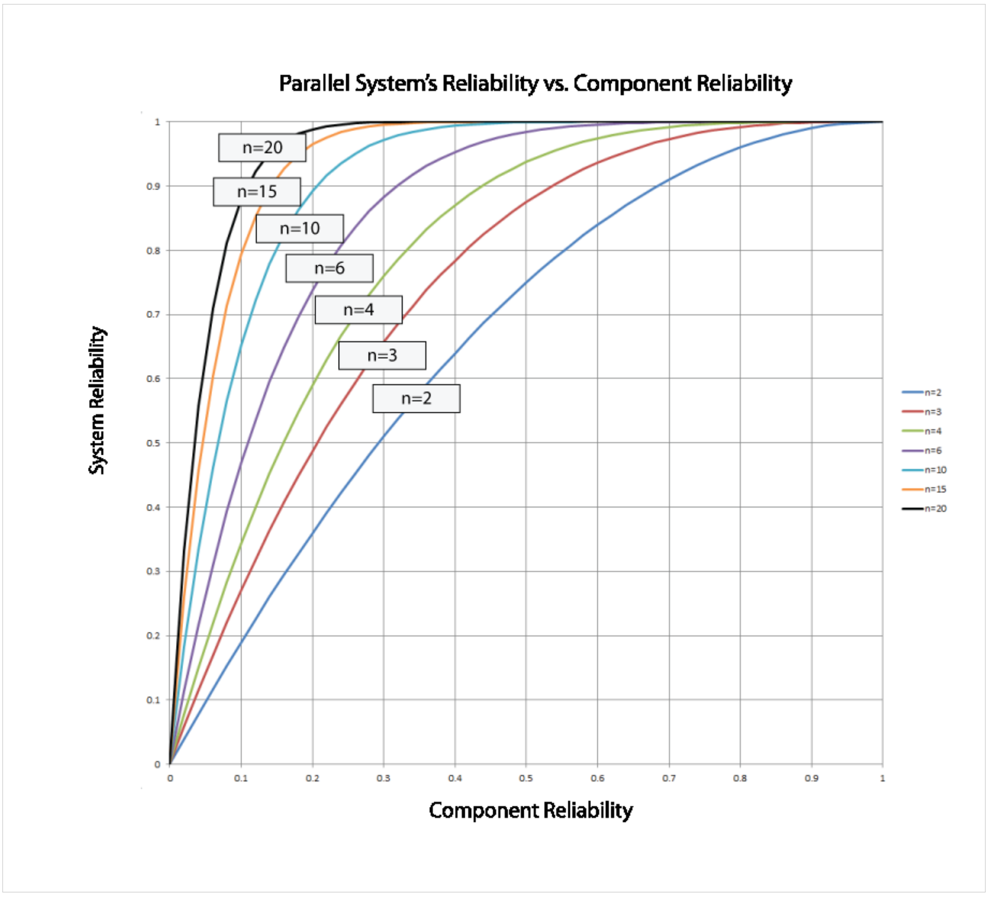
This conclusion can also be illustrated graphically, as shown in the following plot.



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.

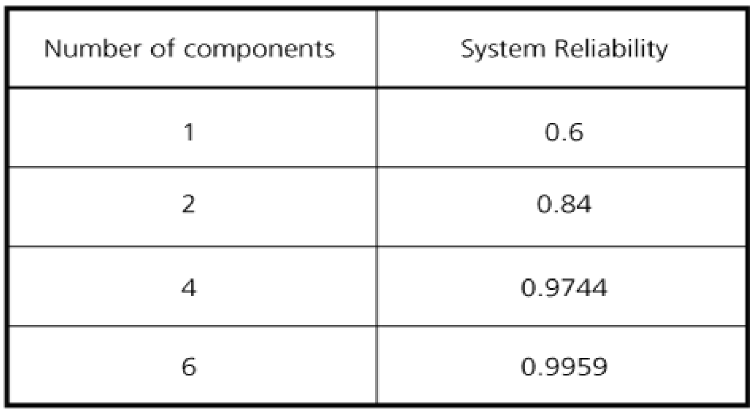
The following plot illustrates that a high system reliability can be achieved with low-reliability components, provided that there are a sufficient number of components in parallel. Note that this plot is the mirror image of the one above that presents the effect of the number of components in a series configuration.



Example: Effect of the Number of Components in a Parallel System

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?

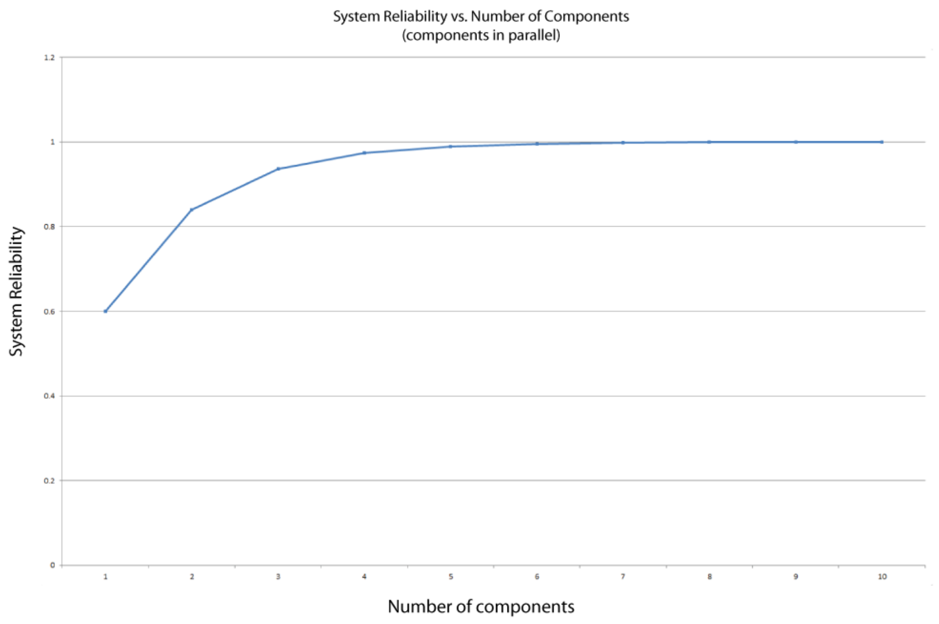
Table 4: System reliability as a function of the number of components.



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

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.

Reliability optimization and costs are covered in detail in Component Reliability Importance.



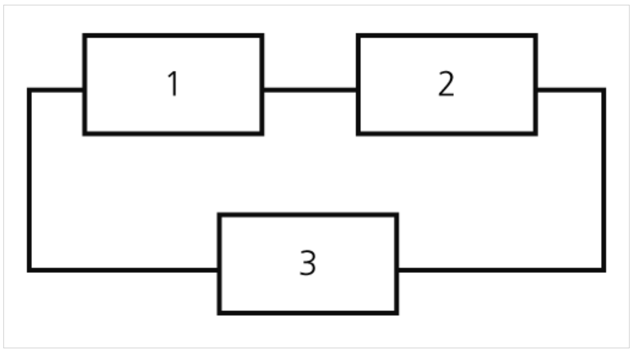
Effect of the number of components in a parallel configuration

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: Calculating the Reliability for a Combination of Series and Parallel

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 $R_1 = 99.5\%$, $R_2 = 98.7\%$and $R_3 = 97.3\%$at 100 hours?

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

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

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:

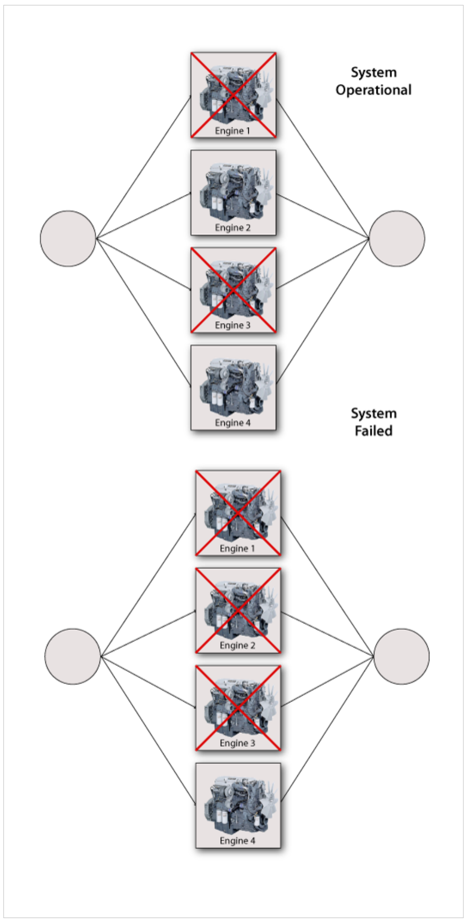
$$\begin{aligned}R_s =&1 - [(1 - 0.982065) \cdot (1 - 0.973000)]\\ R_s =&1 - 0.000484245\\ R_s =&0.999515755\\ R_s =&99.95\%\end{aligned}$$

k-out-of-n Parallel Configuration

The *k*-out-of- *n* configuration is a special case of parallel redundancy. This type of configuration requires that at least $k$components succeed out of the total $n$parallel components for the system to succeed. For example, consider an airplane that has four engines. Furthermore, suppose that the design of the aircraft is such that at least two engines are required to function for the aircraft to remain airborne. This means that the engines are reliability-wise in a *k*-out-of- *n* configuration, where *k* = 2 and *n* = 4. More specifically, they are in a 2-out-of-4 configuration.

Even though we classified the *k*-out-of-*n* configuration as a special case of parallel redundancy, it can also be viewed as a general configuration type. As the number of units required to keep the system functioning approaches the total number of units in the system, the system's behavior tends towards that of a series system. If the number of units required is equal to the number of units in the system, it is a series system. In other words, a series system of statistically independent components is an *n*-out-of-*n* system and a parallel system of statistically independent components is a 1-out-of-*n* system.

Reliability of *k*-out-of-*n* Independent and Identical Components



The simplest case of components in a *k*-out-of-*n* configuration is when the components are independent and identical. In other words, all the components have the same failure distribution and whenever a failure occurs, the remaining components are not affected. In this case, the reliability of the system with such a configuration can be evaluated using the binomial distribution, or:

$$R_s(k,n,R) = \sum_{r=k}^{n} \binom{n}{r} R^r (1-R)^{n-r}$$

where:

- *n* is the total number of units in parallel.
- *k* is the minimum number of units required for system success.
- *R* is the reliability of each unit.

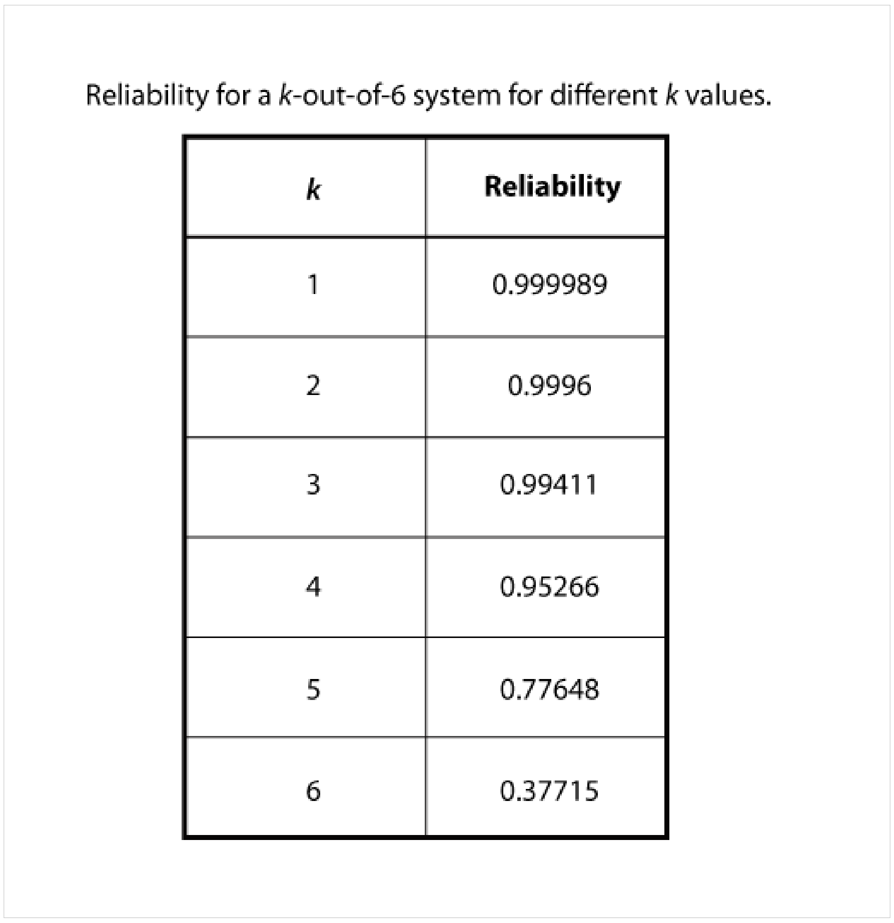
Example: Calculating the Reliability for a *k*-out-of-*n* System

Consider a system of 6 pumps of which at least 4 must function properly for system success. Each pump has an 85% reliability for the mission duration. What is the probability of success of the system for the same mission duration?

Using $R_s(k,n,R) = \sum_{r=k}^{n} \binom{n}{r} R^r (1-R)^{n-r}$ for $k=4$ and $n=6$:

$$\begin{aligned} R_s &= \sum_{r=4}^{6} \binom{6}{r} 0.85^r (1-0.85)^{6-r} \\ &= \binom{6}{4} 0.85^4 (1-0.85)^2 + \binom{6}{5} 0.85^5 (1-0.85)^1 \\ &\quad + \binom{6}{6} 0.85^6 (1-0.85)^0 \\ &= 0.1762 + 0.3993 + 0.3771 \\ &= 95.26\% \end{aligned}$$

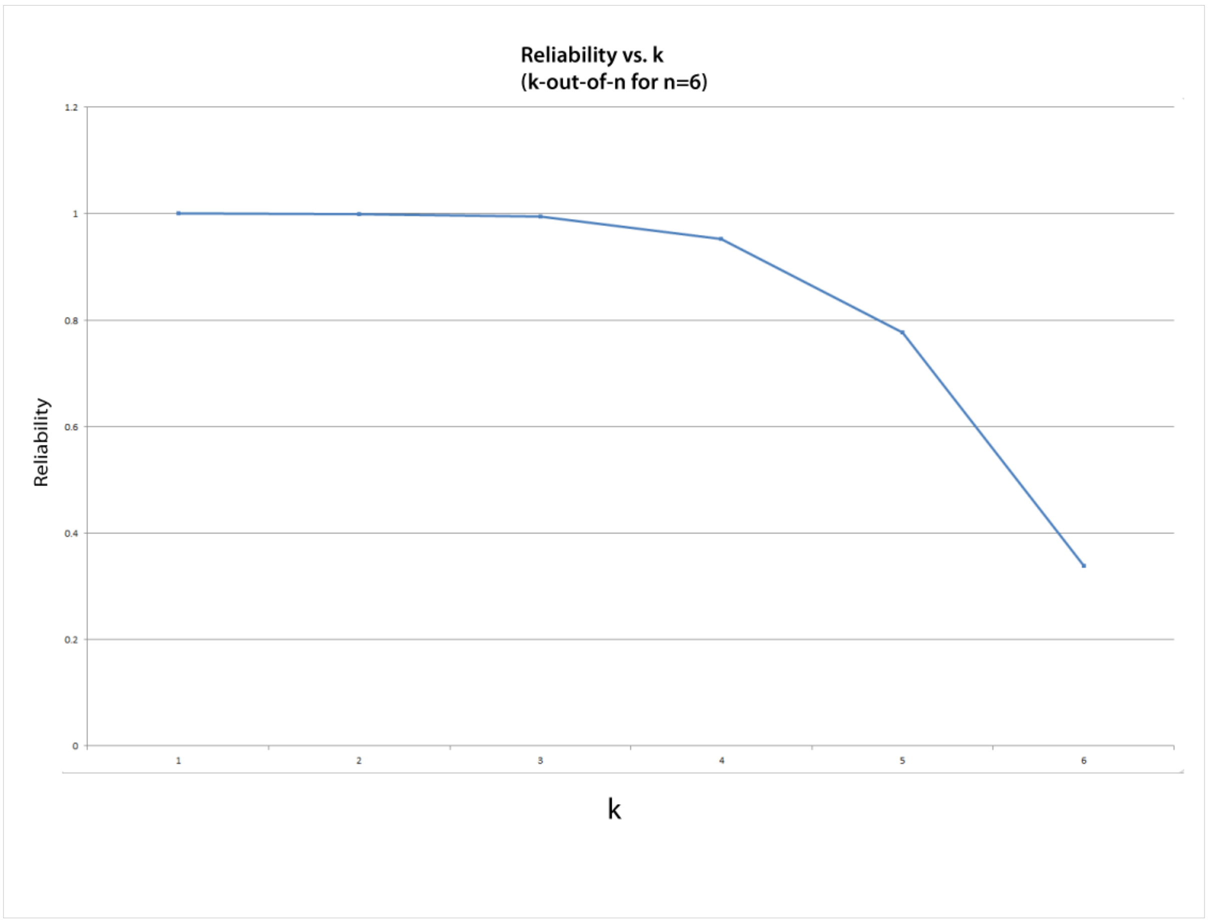
One can examine the effect of increasing the number of units required for system success while the total number of units remains constant (in this example, six units). In the figure above, the reliability of the *k*-out-of-6 configuration was plotted versus different numbers of required units.



Reliability for a *k*-out-of-6 system for different *k* values.

k	**Reliability**
1	0.999989
2	0.9996
3	0.99411
4	0.95266
5	0.77648
6	0.37715

Note that the system configuration becomes a simple parallel configuration for *k* = 1 and the system is a six-unit series configuration $((0.85)^6 = 0.377)$ for $k = 6$.



Reliability of Nonidentical *k*-out-of-*n* Independent Components

In the case where the *k*-out-of-*n* components are not identical, the reliability must be calculated in a different way. One approach, described in detail later in this chapter, is to use the event space method. In this method, all possible operational combinations are considered in order to obtain the system's reliability. The method is illustrated with the following example.

Example: Calculating Reliability for *k*-out-of-*n* If Components Are Not Identical

Three hard drives in a computer system are configured reliability-wise in parallel. At least two of them must function in order for the computer to work properly. Each hard drive is of the same size and speed, but they are made by different manufacturers and have different reliabilities. The reliability of HD #1 is 0.9, HD #2 is 0.88 and HD #3 is 0.85, all at the same mission time.

Since at least two hard drives must be functioning at all times, only one failure is allowed. This is a 2-out-of-3 configuration.

The following operational combinations are possible for system success:

1. All 3 hard drives operate.
2. HD #1 fails while HDs #2 and #3 continue to operate.
3. HD #2 fails while HDs #1 and #3 continue to operate.
4. HD #3 fails while HDs #1 and #2 continue to operate.

The probability of success for the system (reliability) can now be expressed as:

$$\begin{aligned} P_s = & R_1R_2R_3 + (1-R_1)R_2R_3 + R_1(1-R_2)R_3 \\ & + R_1R_2(1-R_3) \end{aligned}$$

This equation for the reliability of the system can be reduced to:

$$R_s = R_1R_2 + R_2R_3 + R_1R_3 - 2R_1R_2R_3$$

or:

$$R_s = 95.86\%$$

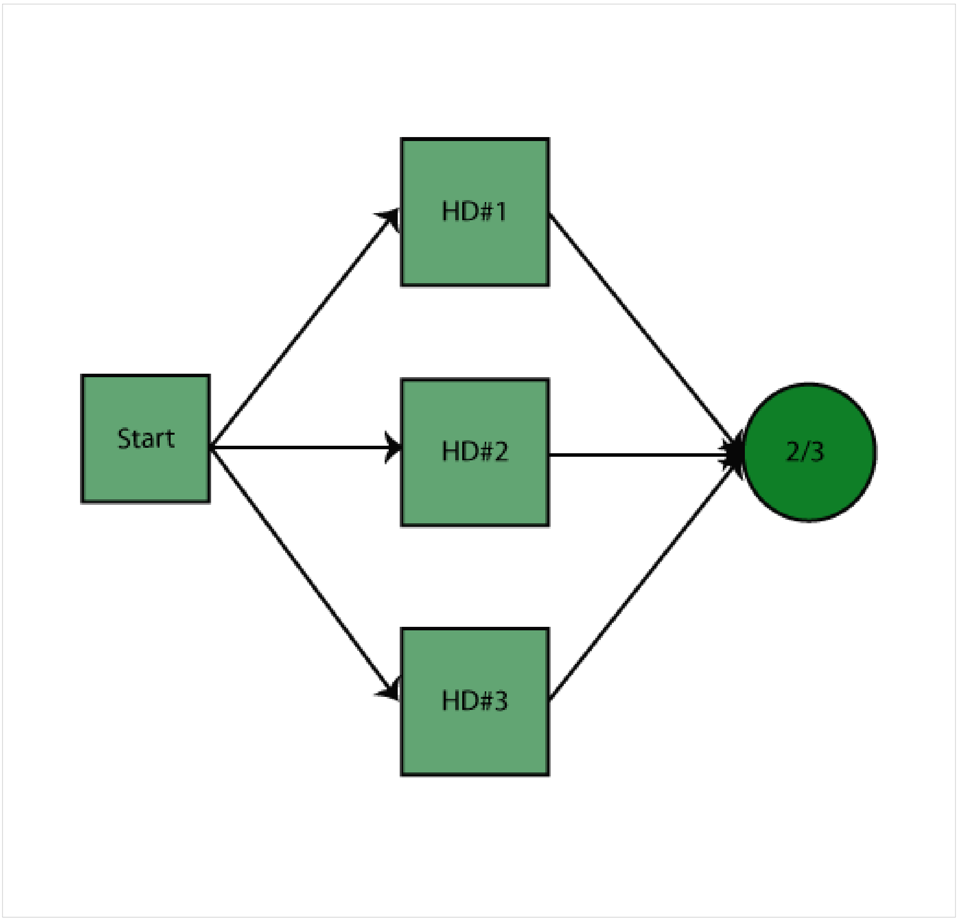
If all three hard drives had the same reliability, R, then the equation for the reliability of the system could be further reduced to:

$$R_s = 3R^2 - 2R^3$$

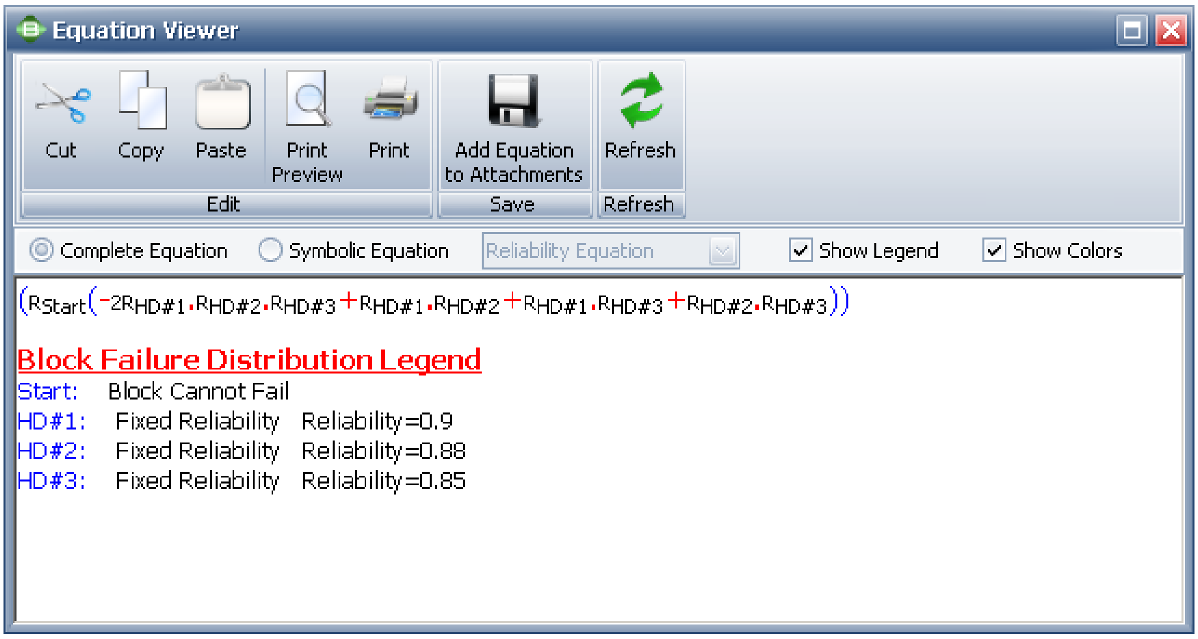
Or, using the binomial approach:

$$\begin{aligned} R_s =& \sum_{r=2}^{3} \binom{3}{r} R^r(1-R)^{3-r} \\ =& \binom{3}{2} R^2(1-R) + \binom{3}{3} R^3(1-R)^0 \\ =& 3 \cdot R^2(1-R) + R^3 \\ =& 3R^2 - 2R^3 \end{aligned}$$

The example can be repeated using BlockSim. The following graphic demonstrates the RBD for the system.



The RBD is analyzed and the system reliability equation is returned. The following figure shows the equation returned by BlockSim.

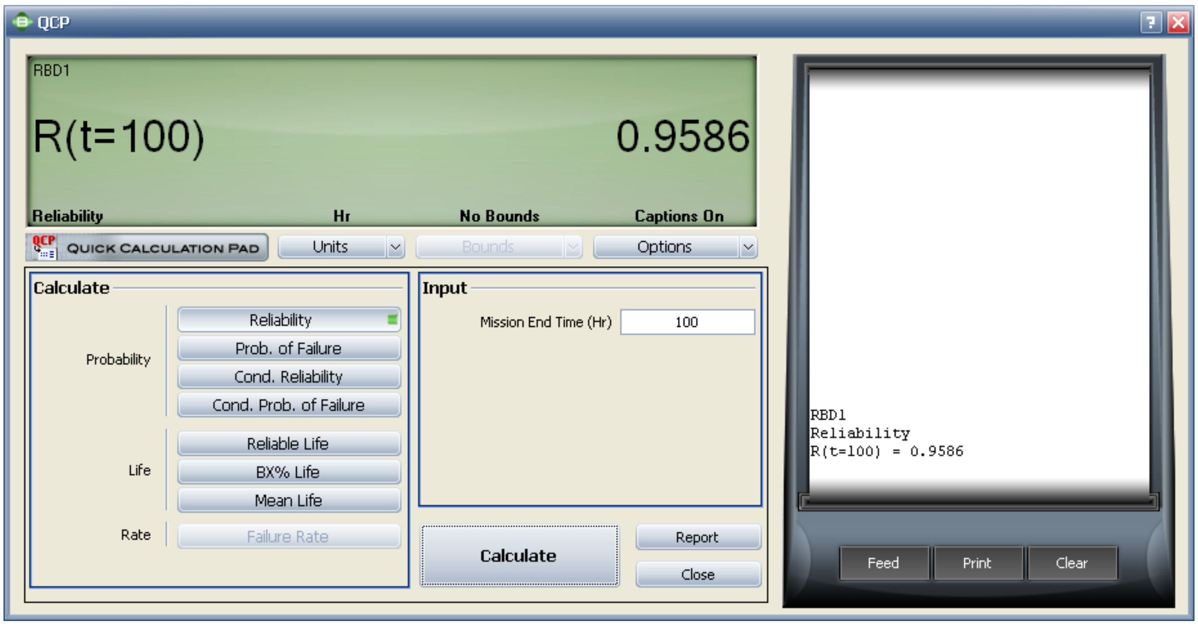


$(R_{Start}(-2R_{HD\#1} \cdot R_{HD\#2} \cdot R_{HD\#3} + R_{HD\#1} \cdot R_{HD\#2} + R_{HD\#1} \cdot R_{HD\#3} + R_{HD\#2} \cdot R_{HD\#3}))$

Block Failure Distribution Legend

Start:	Block Cannot Fail	
HD#1:	Fixed Reliability	Reliability=0.9
HD#2:	Fixed Reliability	Reliability=0.88
HD#3:	Fixed Reliability	Reliability=0.85

Using the Analytical Quick Calculation Pad, the reliability can be calculated to be 0.9586. The following figure shows the returned result. Note that you are not required to enter a mission end time for this system into the Analytical QCP because all of the components are static and thus the reliability results are independent of time.



RBD1

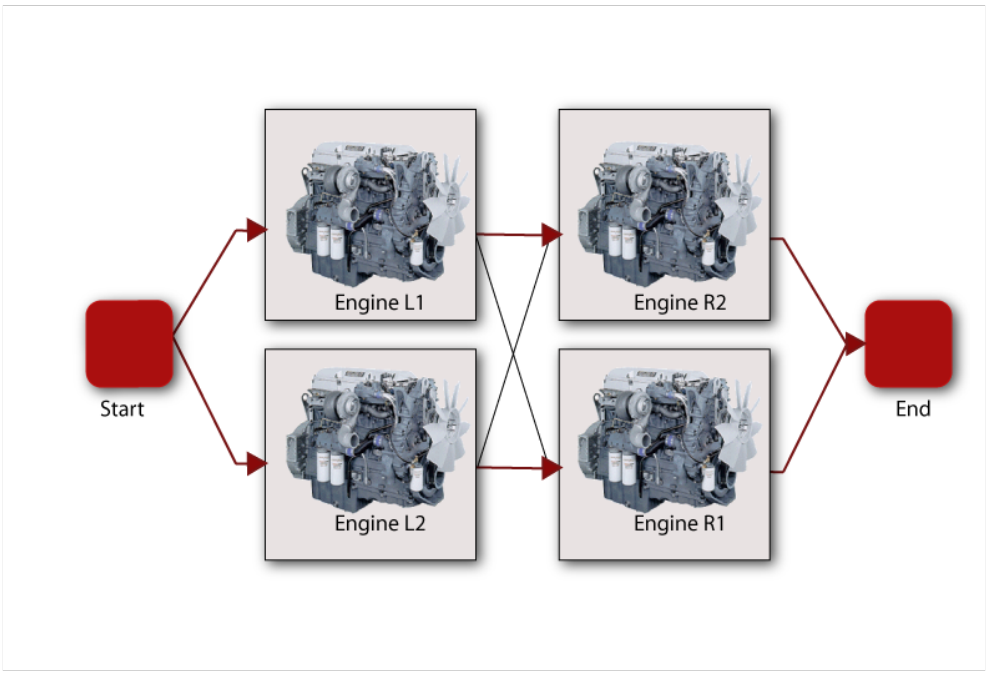
R(t=100) 0.9586

Mission End Time (Hr) 100

```
RBD1
Reliability
R(t=100) = 0.9586
```

Example 10

Consider the four-engine aircraft discussed previously. If we were to change the problem statement to two out of four engines are required, however no two engines on the same side may fail, then the block diagram would change to the configuration shown below.

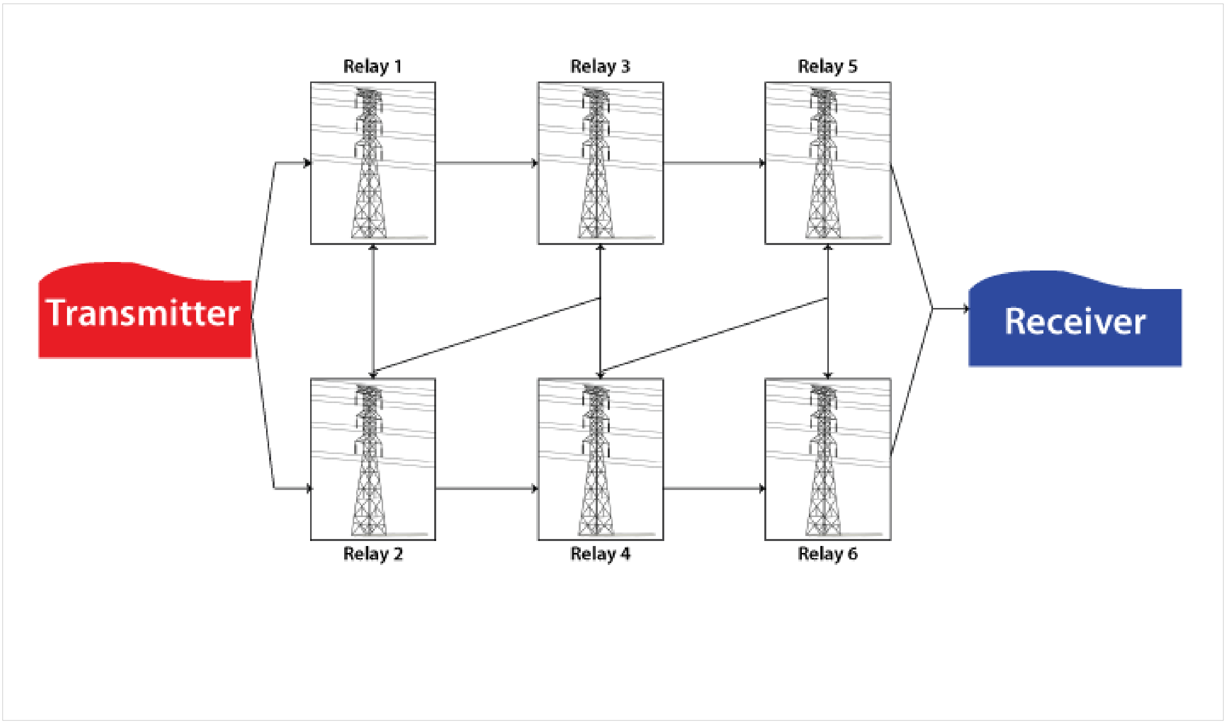


Note that this is the same as having two engines in parallel on each wing and then putting the two wings in series.

Consecutive *k*-out-of-*n* : *F* Redundancy

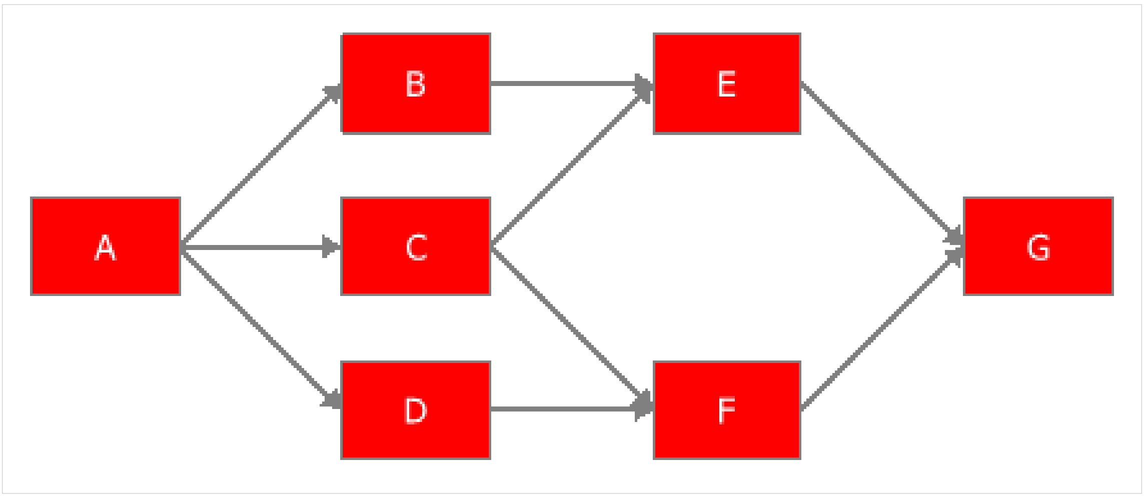
There are other multiple redundancy types and multiple industry terms. One such example is what is referred to as a consecutive k-out-of-n : F system. To illustrate this configuration type, consider a telecommunications system that consists of a transmitter and receiver with six relay stations to connect them. The relays are situated so that the signal originating from one station can be picked up by the next two stations down the line. For example, a signal from the transmitter can be received by Relay 1 and Relay 2, a signal from Relay 1 can be received by Relay 2 and Relay 3, and so forth. Thus, this arrangement would require two consecutive relays to fail for the system to fail. A diagram of this configuration is shown in the following figure.

This type of a configuration is also referred to as a *complex system*. Complex systems are discussed in the next section.



Complex Systems

In many cases, it is not easy to recognize which components are in series and which are in parallel in a complex system. The network shown next is a good example of such a complex system.



The system in the figure above cannot be broken down into a group of series and parallel systems. This is primarily due to the fact that component Chas two paths leading away from it, whereas $B$and Dhave only one. Several methods exist for obtaining the reliability of a complex system including:

- The decomposition method.
- The event space method.
- The path-tracing method.

Decomposition Method

The decomposition method is an application of the law of total probability. It involves choosing a "key" component and then calculating the reliability of the system twice: once as if the key component failed ($R=0$) and once as if the key component succeeded $(R=1)$. These two probabilities are then combined to obtain the reliability of the system, since at any given time the key component will be failed or operating. Using probability theory, the equation is:

$$\begin{aligned} R_s =& P(s\cap A)+P(s\cap \overline{A}) \\ =& P(s|A)P(A)+P(s|\overline{A})P(\overline{A}) \end{aligned}$$

Illustration of the Decomposition Method

Consider three units in series.

- $A$is the event of Unit 1 success.
- $B$is the event of Unit 2 success.
- $C$is the event of Unit 3 success.
- $s$is the event of system success.

First, select a "key" component for the system. Selecting Unit 1, the probability of success of the system is:

$$R_s=P(s|A)P(A)+P(s|\overline{A})P(\overline{A})$$

If Unit 1 is good, then:

$$P(s|A)=R_2R_3$$

That is, if Unit 1 is operating, the probability of the success of the system is the probability of Units 2 and 3 succeeding.

If Unit 1 fails, then:

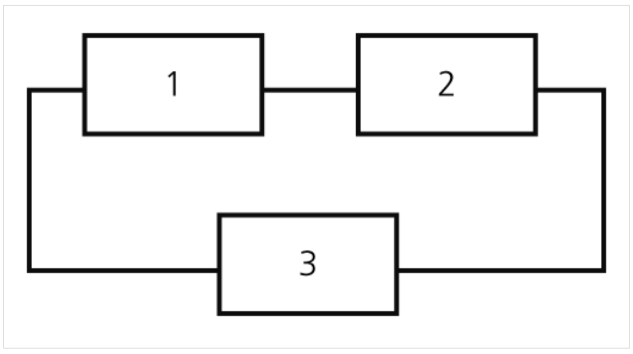
$$P(s|\overline{A})=0$$

That is, if Unit 1 is not operating, the system has failed since a series system requires all of the components to be operating for the system to operate. Thus the reliability of the system is:

$$R_s=R_2R_3P(A)=R_1R_2R_3$$

Another Illustration of the Decomposition Method

Consider the following system:



- $A$is the event of Unit 1 success.
- $B$is the event of Unit 2 success.
- $C$is the event of Unit 3 success.
- $s$is the event of system success.

Selecting Unit 3 as the *key* component, the system reliability is:

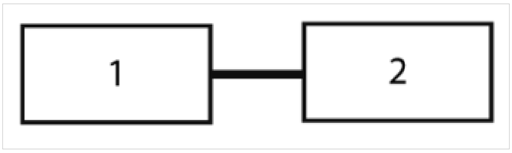
$$R_s=P(s|C)P(C)+P(s|\overline{C})P(\overline{C})$$

If Unit 3 survives, then:

$$P(s|C) = 1$$

That is, since Unit 3 represents half of the parallel section of the system, then as long as it is operating, the entire system operates.

If Unit 3 fails, then the system is reduced to:



$$P(s|\overline{C}) = R_1R_2$$

The reliability of the system is given by:

$$R_s = P(C) + R_1R_2P(\overline{C}) = R_3 + R_1R_2(1 - R_3)$$

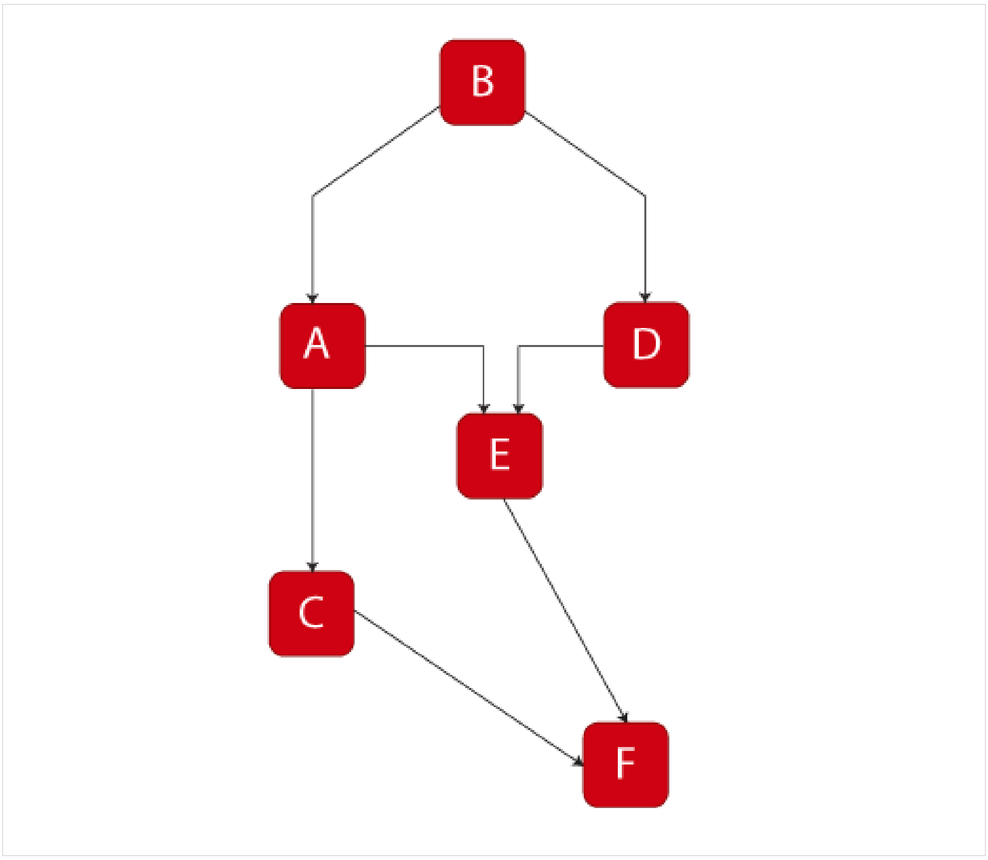
or:

$$R_s = R_3 + R_1R_2 - R_1R_2R_3$$

Example: Using the Decomposition Method to Determine the System Reliability Equation

Do the following for the RBD shown below:

- Use the decomposition method to determine the reliability equation of the system.
- Determine the reliability equation of the same system using BlockSim.



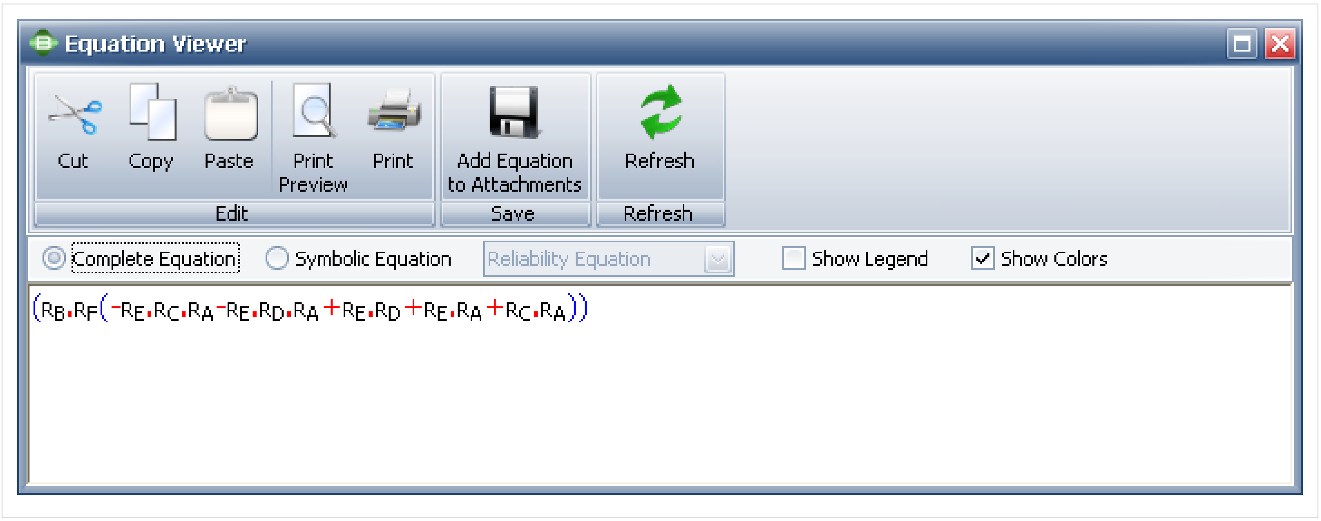
To obtain the solution:

Choose A as the key component, then:

$$\begin{aligned} R_s &= P(s|A)P(A) + P(s|\overline{A})P(\overline{A}) \\ P(s|A) &= R_B R_F \left[1-(1-R_C)(1-R_E)\right] \\ P(s|\overline{A}) &= R_B R_D R_E R_F \\ R_s &= \left[R_B R_F\left[1-(1-R_C)(1-R_E)\right]\right]R_A + \left[R_B R_D R_E R_F\right](1-R_A) \end{aligned}$$

Using BlockSim:

$$R_s = R_B \cdot R_F(-R_A \cdot R_C \cdot R_E - R_A \cdot R_D \cdot R_E + R_A \cdot R_C + R_A \cdot R_E + R_D \cdot R_E)$$



Complete Equation [✓] Symbolic Equation [ ] Show Legend [ ] Show Colors [✓]

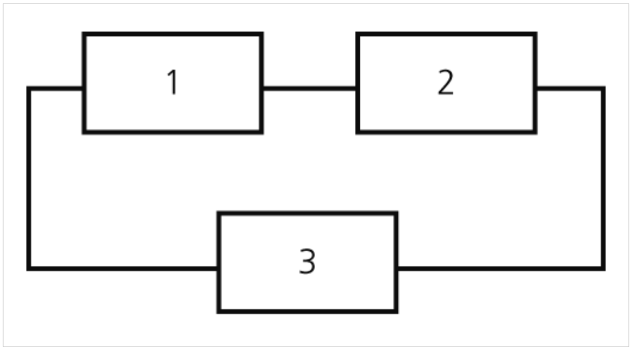
$(R_B \cdot R_F(-R_E \cdot R_C \cdot R_A - R_E \cdot R_D \cdot R_A + R_E \cdot R_D + R_E \cdot R_A + R_C \cdot R_A))$

Event Space Method

The event space method is an application of the mutually exclusive events axiom. All mutually exclusive events are determined and those that result in system success are considered. The reliability of the system is simply the probability of the union of all mutually exclusive events that yield a system success. Similarly, the unreliability is the probability of the union of all mutually exclusive events that yield a system failure. This is illustrated in the following example.

Illustration of the Event Space Method

Consider the following system, with reliabilities R1, R2 and R3 for a given time.



- $A$is the event of Unit 1 success.
- $B$is the event of Unit 2 success.
- $C$is the event of Unit 3 success.

The mutually exclusive system events are:

$$X1 = ABC - \text{all units succeed.}$$
$$X2 = \overline{A}BC - \text{only Unit 1 fails.}$$
$$X3 = A\overline{B}C - \text{only Unit 2 fails.}$$
$$X4 = AB\overline{C} - \text{only Unit 3 fails.}$$
$$X5 = \overline{AB}C - \text{Units 1 and 2 fail.}$$
$$X6 = \overline{A}B\overline{C} - \text{Units 1 and 3 fail.}$$
$$X7 = A\overline{BC} - \text{Units 2 and 3 fail.}$$
$$X8 = \overline{ABC} - \text{all units fail.}$$

System events X_6, $X_7$and X_8result in system failure. Thus the probability of failure of the system is:

$$P_f = P(X_6 \cup X_7 \cup X_8)$$

Since events X_6, $X_7$and $X_8$are mutually exclusive, then:

$$P_f = P(X_6) + P(X_7) + P(X_8)$$

and:

$$P(X_6) = P(\overline{A}B\overline{C}) = (1 - R_1)(R_2)(1 - R_3)$$
$$P(X_7) = P(A\overline{BC}) = (R_1)(1 - R_2)(1 - R_3)$$
$$P(X_8) = P(\overline{ABC}) = (1 - R_1)(1 - R_2)(1 - R_3)$$

Combining terms yields:

$$P_f = 1 - R_1R_2 - R_3 + R_1R_2R_3$$

Since:

$$R_s = 1 - P_f$$

Then:

$$R_s = R_1R_2 + R_3 - R_1R_2R_3$$

This is of course the same result as the one obtained previously using the decomposition method. If $R_1 = 99.5\%$, $R_2 = 98.7\%$and $R_3 = 97.3\%$, then:

$$\begin{aligned} R_s &= 0.995 \cdot 0.987 + 0.973 - 0.995 \cdot 0.987 \cdot 0.973 \\ &= 0.999515755 \end{aligned}$$

or:

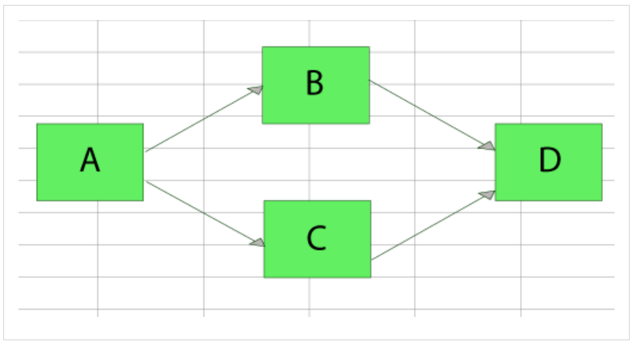
$$R_s = 99.95\%$$

Path-Tracing Method

With the path-tracing method, every path from a starting point to an ending point is considered. Since system success involves having at least one path available from one end of the RBD to the other, as long as at least one path from the beginning to the end of the path is available, then the system has not failed. One could consider the RBD to be a plumbing schematic. If a component in the system fails, the "water" can no longer flow through it. As long as there is at least one path for the "water" to flow from the start to the end of the system, the system is successful. This method involves identifying all of the paths the "water" could take and calculating the reliability of the path based on the components that lie along that path. The reliability of the system is simply the probability of the union of these paths. In order to maintain consistency of the analysis, starting and ending blocks for the system must be defined.

Illustration of the Path-Tracing Method

Obtain the reliability equation of the following system.



The successful paths for this system are:

$$X_1 = ABD \text{ and } X_2 = ACD$$

The reliability of the system is simply the probability of the union of these paths:

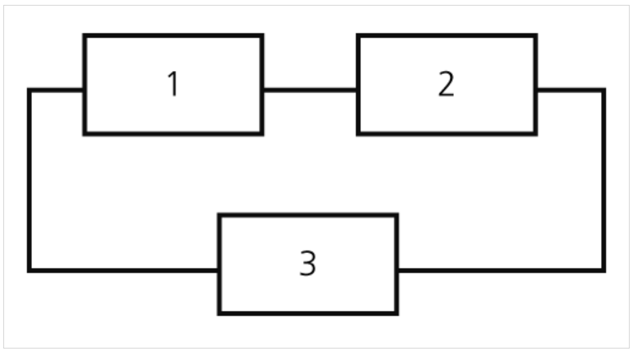
$$\begin{aligned} R_s &= P(X_1 \cup X_2) \\ P(X_1 \cup X_2) &= P(X_1) + P(X_2) - P(X_1 \cap X_2) \\ &= P(ABD) + P(ACD) - P(ABCD) \end{aligned}$$

Thus:

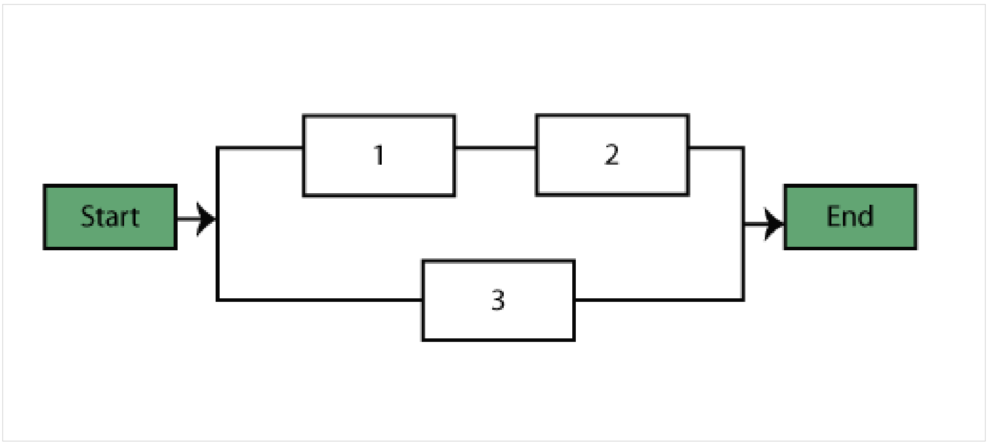
$$R_s = R_A R_B R_D + R_A R_C R_D - R_A R_B R_C R_D$$

Another Illustration of the Path-Tracing Method

Obtain the reliability equation of the following system.



Assume starting and ending blocks that cannot fail, as shown next.



The paths for this system are:

$$X_1 = 1, 2 \text{ and } X_2 = 3$$

The probability of success of the system is given by:

$$\begin{aligned} P(X_1 \cup X_2) =& P(X_1) + P(X_2) - P(X_1 \cap X_2) \\ =& P(1,2) + P(3) - P(1,2,3) \end{aligned}$$

or:

$$R_s = R_1 R_2 + R_3 - R_1 R_2 R_3$$

Starting and Ending Blocks in BlockSim

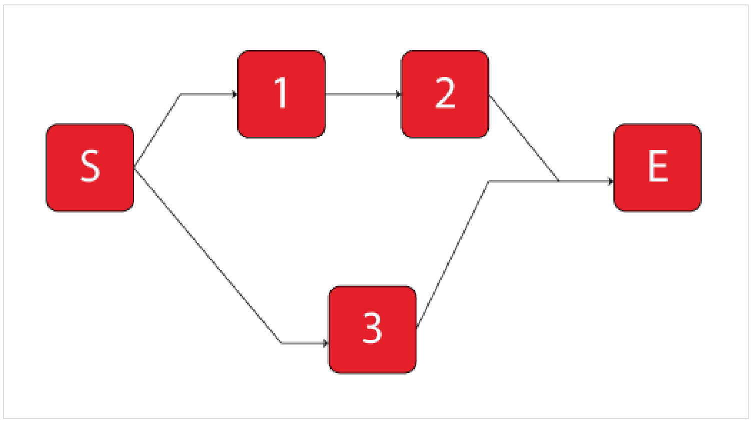
Note that BlockSim requires that all diagrams start from a single block and end on a single block. To meet this requirement for this example, we arbitrarily added a starting and an ending block, as shown in the diagram below. These blocks can be set to a cannot fail condition, or $R = 1$, and thus not affect the outcome. However, when the analysis is performed in BlockSim, the returned equation will include terms for the non-failing blocks, as shown in the picture of the Equation Viewer.

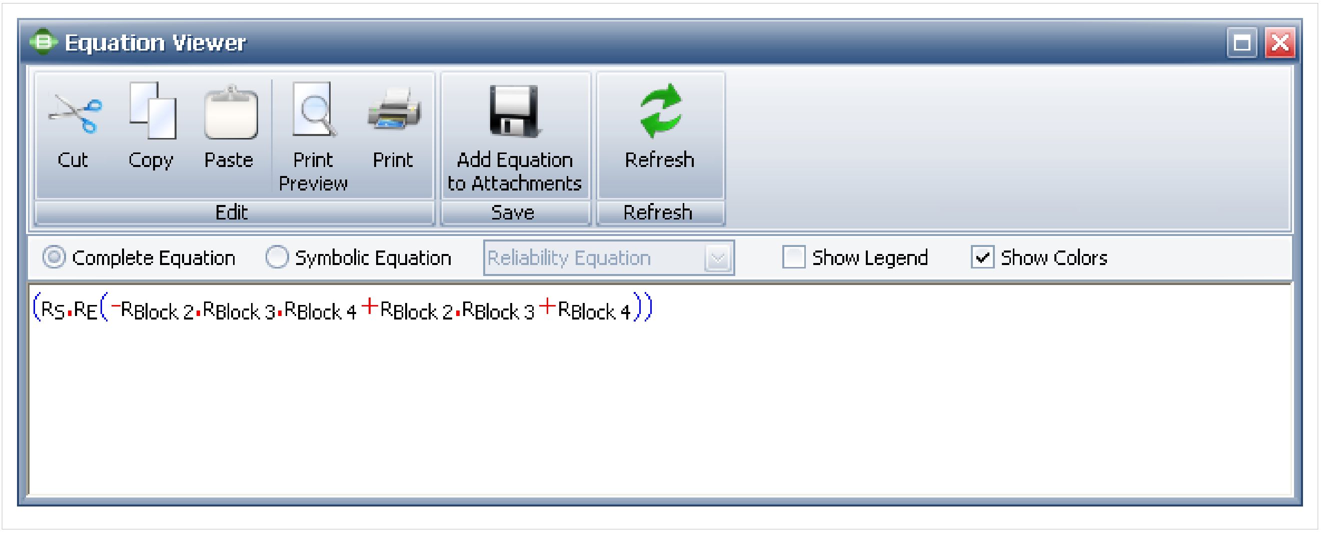
$$R_{system} = (R_S \cdot R_E(-R_1 \cdot R_2 \cdot R_3 + R_1 \cdot R_2 + R_3))$$

Note that since $R_S = R_E = 1$, the system equation above, can be reduced to:

$$\begin{aligned} R_{system} =& (1 \cdot 1(-R_1 \cdot R_2 \cdot R_3 + R_1 \cdot R_2 + R_3)) \\ =& -R_1 \cdot R_2 \cdot R_3 + R_1 \cdot R_2 + R_3 \end{aligned}$$

This is equivalent to $R_s = R_1 R_2 + R_3 - R_1 R_2 R_3$. The reason that BlockSim includes all items regardless of whether they can fail or not is because BlockSim only recomputes the equation when the system structure has changed. What this means is that the user can alter the failure characteristics of an item without altering the diagram structure. For example, a block that was originally set not to fail can be re-set to a failure distribution and thus it would need to be used in subsequent analyses.





[✓ Complete Equation] Symbolic Equation

Show Legend [✓ Show Colors]

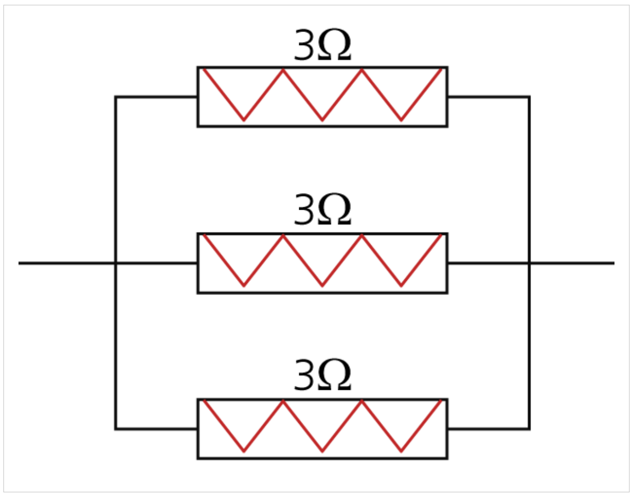
(RS·RE(-RBlock 2·RBlock 3·RBlock 4+RBlock 2·RBlock 3+RBlock 4))

Difference Between Physical and Reliability-Wise Arrangement

Reliability block diagrams are created in order to illustrate the way that components are arranged reliability-wise in a system. So far we have described possible structural properties of a system of components, such as series, parallel, etc. These structural properties, however, refer to the system's state of success or failure based on the states of its components. The physical structural arrangement, even though clearly related to the reliability-wise arrangement, is not necessarily identical to it.

Example: Physical vs. Reliability-Wise Arrangement

Consider the following circuit:



The equivalent resistance must always be less than 1.2Ω.

Draw the reliability block diagram for this circuit.

First, let's consider the case where all three resistors operate:

$$\begin{aligned}\frac{1}{r_{eq}} &= \frac{1}{r_1}+\frac{1}{r_2}+\frac{1}{r_3}\\ &= \frac{1}{3}+\frac{1}{3}+\frac{1}{3}\\ &= 1\Omega\end{aligned}$$

Thus, when all components operate, the equivalent resistance is 1Ω, which is less than the maximum resistance of 1.2Ω.

Next, consider the case where one of the resistors fails open. In this case, the resistance for the resistor is infinite and the equivalent resistance is:

$$\frac{1}{r_{eq}} = \frac{1}{\infty}+\frac{1}{3}+\frac{1}{3}=\frac{2}{3}$$

Thus:

$$r_{eq} = 1.5\Omega > 1.2\Omega \text{ - System failed}$$

If two resistors fail open (e.g., #1 and #2), the equivalent resistance is:

$$\frac{1}{r_{eq}} = \frac{1}{\infty}+\frac{1}{\infty}+\frac{1}{3}=\frac{1}{3}$$

Thus:

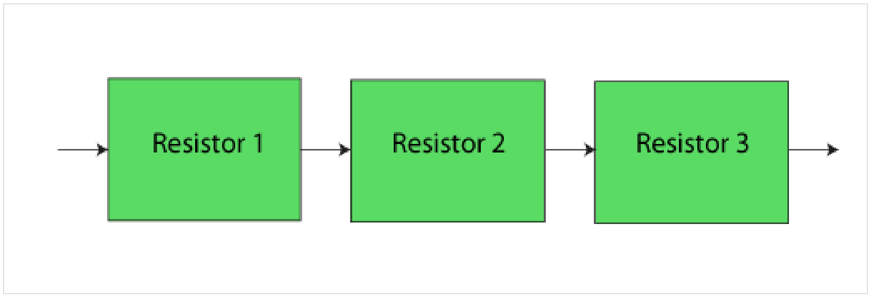
$$r_{eq} = 3\Omega > 1.2\Omega \text{ - System failed}$$

If all three resistors fail open:

$$r_{eq} = \infty > 1.2\Omega \text{ - System failed}$$

Thus, if r_1, r_2, r_3, or any combination of the three fails, the system fails. Put another way, $r_1$and $r_2$and $r_3$must succeed in order for the system to succeed.

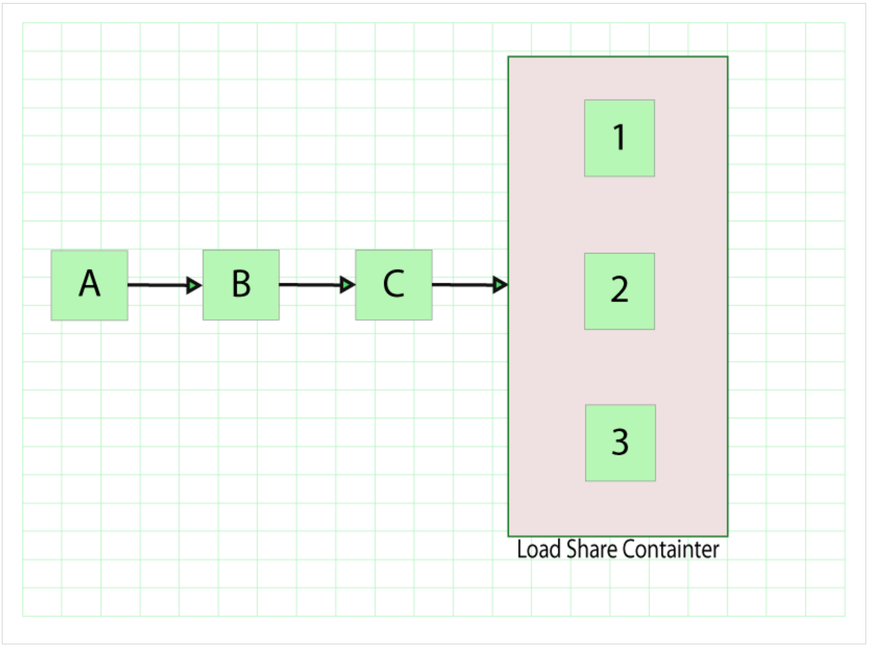
The RBD is:



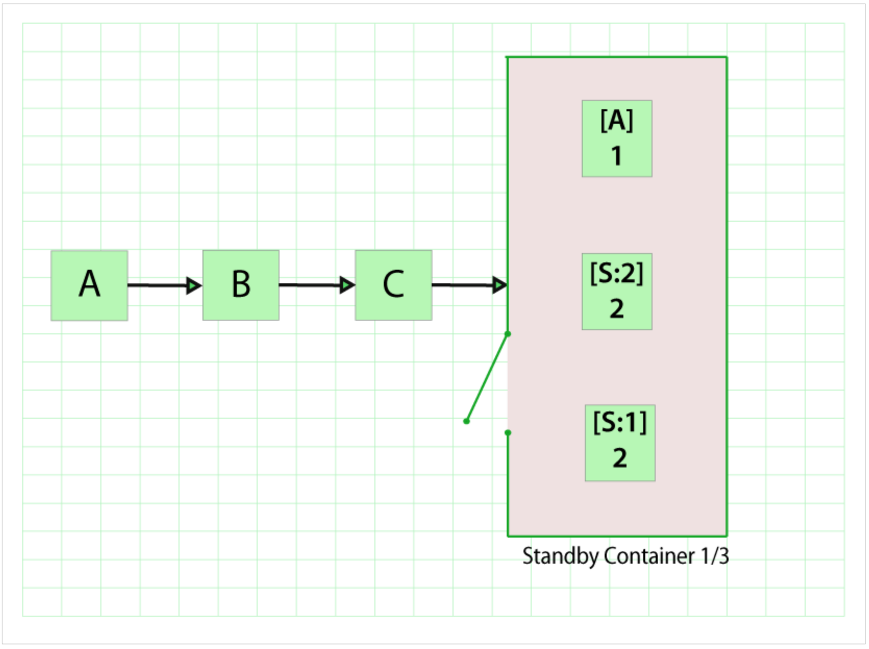
In this example it can be seen that even though the three components were physically arranged in parallel, their reliability-wise arrangement is in series.

Load Sharing and Standby Redundancy

Units in load sharing redundancy exhibit different failure characteristics when one or more fail. In the figure below, blocks 1, 2 and 3 are in a load sharing container in BlockSim and have their own failure characteristics. All three must fail for the container to fail. However, as individual items fail, the failure characteristics of the remaining units change since they now have to carry a higher load to compensate for the failed ones. The failure characteristics of each block in a load sharing container are defined using both a life distribution and a life-stress relationship that describe how the life distribution changes as the load changes.



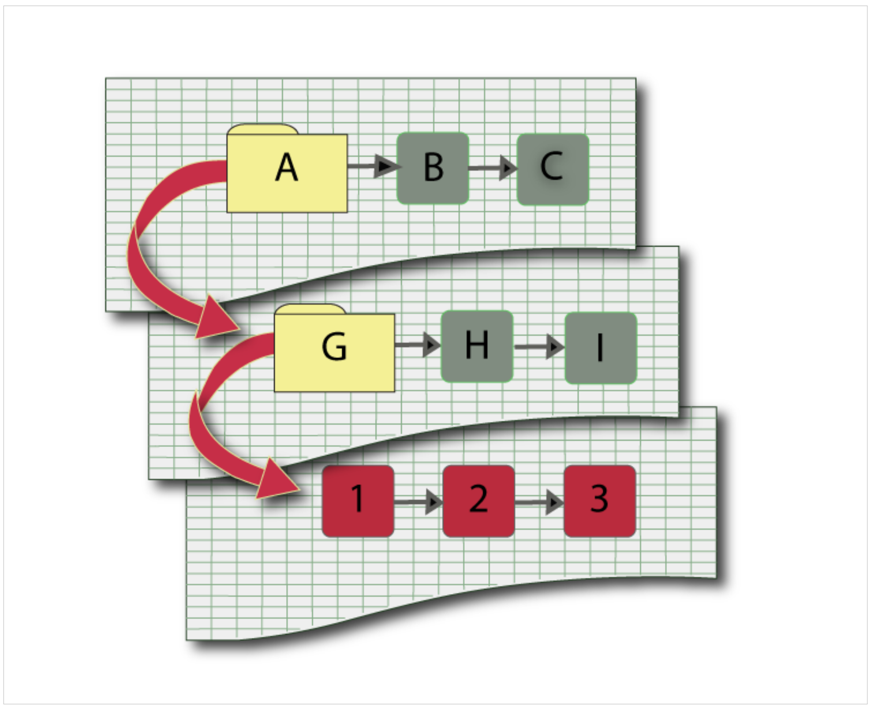
Standby redundancy configurations consist of items that are inactive and available to be called into service when/if an active item fails (i.e., the items are on standby). Within BlockSim, a container block with other blocks inside is used to better achieve and streamline the representation and analysis of standby configurations. The container serves a dual purpose. The first is to clearly delineate and define the standby relationships between the active unit(s) and standby unit(s). The second is to serve as the manager of the switching process. For this purpose, the container can be defined with its own probability of successfully activating standby units when needed. The next figure includes a standby container with three items in standby configuration where one component is active while the other two components are idle. One block within the container must be operating, otherwise the container will fail, leading to a system failure (since the container block is part of a series configuration in this example).



Because both of these concepts are better understood when time dependency is considered, they are addressed in more detail in Time-Dependent System Reliability (Analytical).

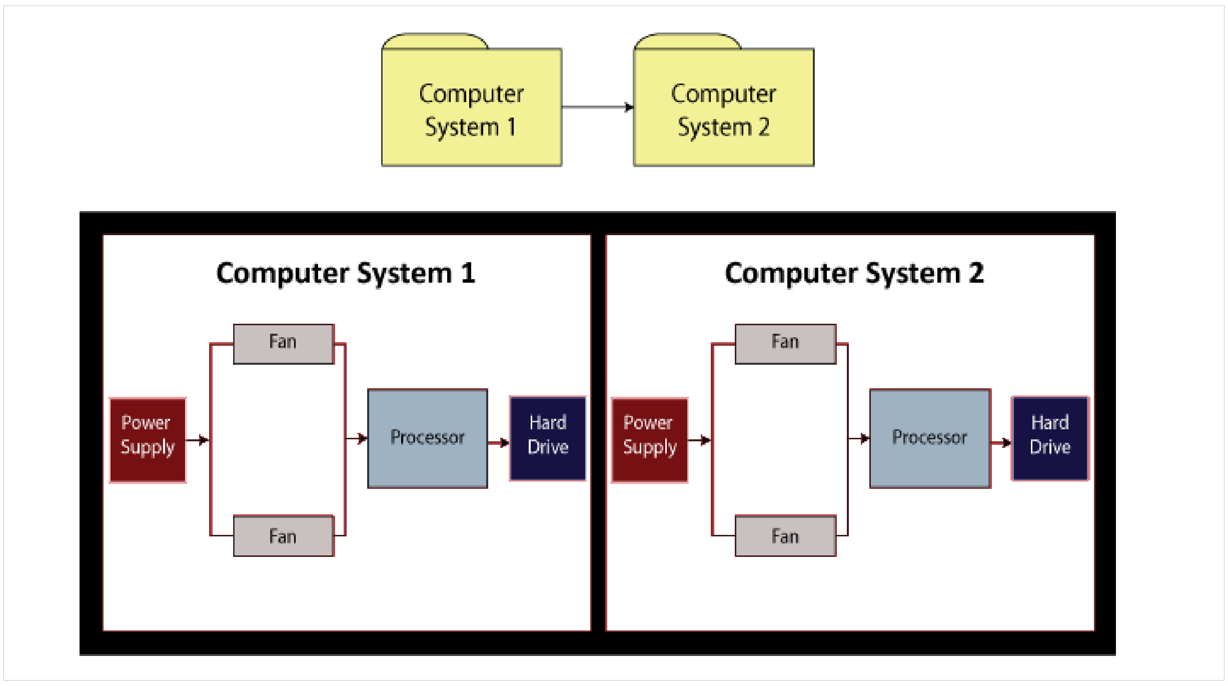
Inherited Subdiagrams

Within BlockSim, a subdiagram block inherits some or all of its properties from another block diagram. This allows the analyst to maintain separate diagrams for portions of a system and incorporate those diagrams as components of another diagram. With this technique, it is possible to generate and analyze extremely complex diagrams representing the behavior of many subsystems in a manageable way. In the first figure below, Subdiagram Block A in the top diagram represents the series configuration of the subsystem reflected in the middle diagram, while Subdiagram Block G in the middle diagram represents the series configuration of the subsubsystem in the bottom diagram.



Example: Calculating the Reliability with Subdiagrams

For this example, obtain the reliability equation of the system shown below.



The system reliability equation is:

$$R_{System} = R_{Computer1} \cdot R_{Computer2}$$

Now:

$$R_{Computer1} = (R_{Power\ Supply} \cdot R_{Processor} \cdot R_{HardDrive} \cdot (-R_{Fan} \cdot R_{Fan} + R_{Fan} + R_{Fan}))$$

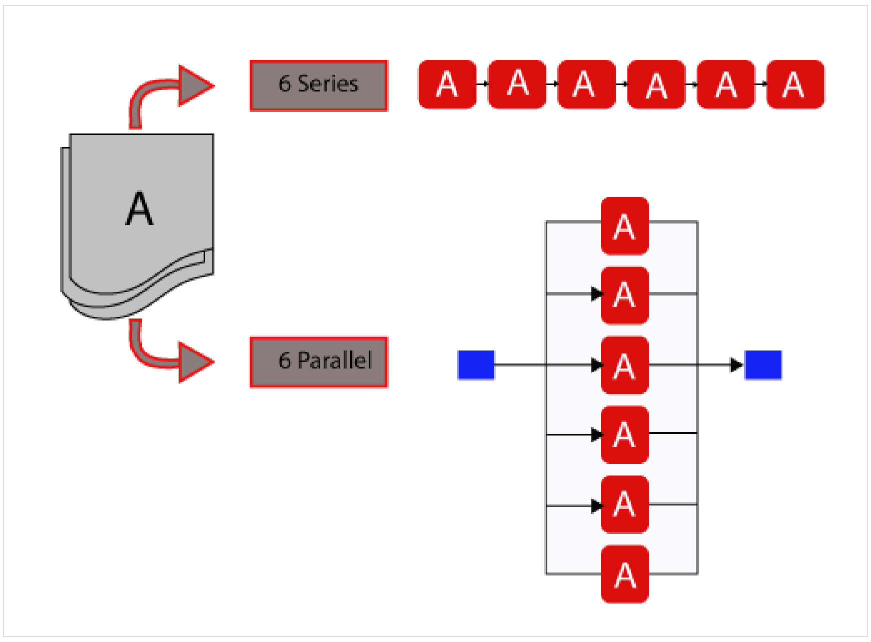
Since the structures of the computer systems are the same, $R_{Computer1} = R_{Computer2}$, then substituting the first equation above into the second equation above yields:

$$R_{System} = (R_{Power\ Supply} \cdot R_{Processor} \cdot R_{HardDrive}(-R_{Fan}^2 + 2R_{Fan}))^2$$

When using BlockSim to compute the equation, the software will return the first equation above for the system and the second equation above for the subdiagram. Even though BlockSim will make these substitutions internally when performing calculations, it does show them in the System Reliability Equation window.

Multi Blocks

By using multi blocks within BlockSim, a single block can represent multiple identical blocks in series or in parallel configuration. This technique is simply a way to save time when creating the RBD and to save space within the diagram. Each item represented by a multi block is a separate entity with identical reliability characteristics to the others. However, each item is not rendered individually within the diagram. In other words, if the RBD contains a multi block that represents three identical components in a series configuration, then each of those components fails according to the same failure distribution but each component may fail at different times. Because the items are arranged reliability-wise in series, if one of those components fails, then the multi block fails. It is also possible to define a multi block with multiple identical components arranged reliability-wise in parallel or *k*-out-of-*n* redundancy. The following figure demonstrates the use of multi blocks in BlockSim.

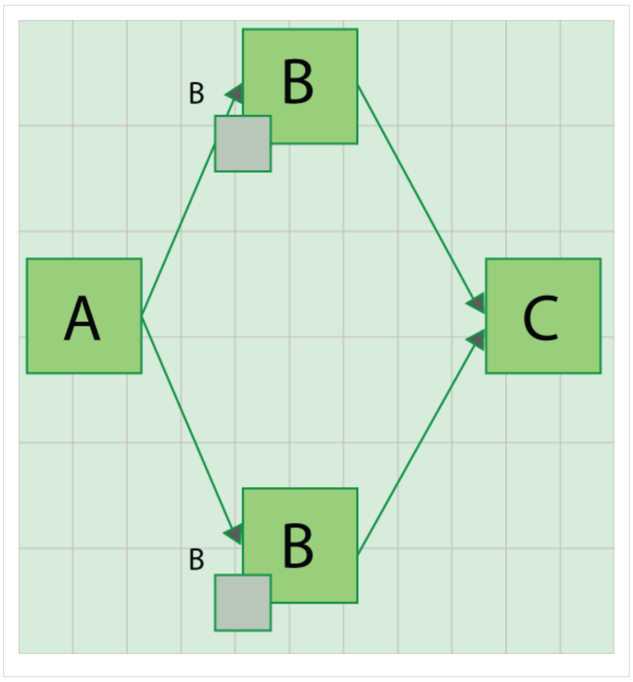


Mirrored Blocks

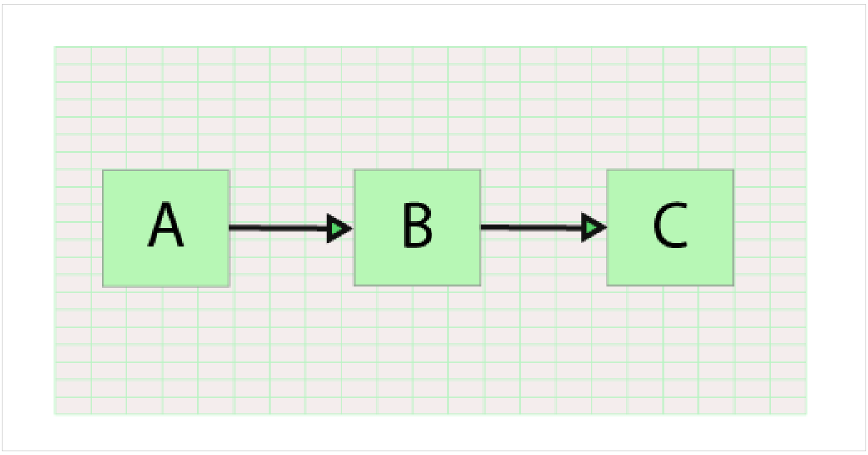
While multi blocks allow the analyst to represent multiple items with a single block in an RBD, BlockSim's mirrored blocks can be used to represent a single item with more than one block placed in multiple locations within the diagram. Mirrored blocks can be used to simulate bidirectional paths within a diagram. For example, in a reliability block diagram for a communications system where the lines can operate in two directions, the use of mirrored blocks will facilitate realistic simulations for the system maintainability and availability.

It may also be appropriate to use this type of block if the component performs more than one function and the failure to perform each function has a different reliability-wise impact on the system. In mirrored blocks, the duplicate block behaves in the exact same way that the original block does. The failure times and all maintenance events are the same for each duplicate block as for the original block.

To better illustrate this consider the following block diagram:

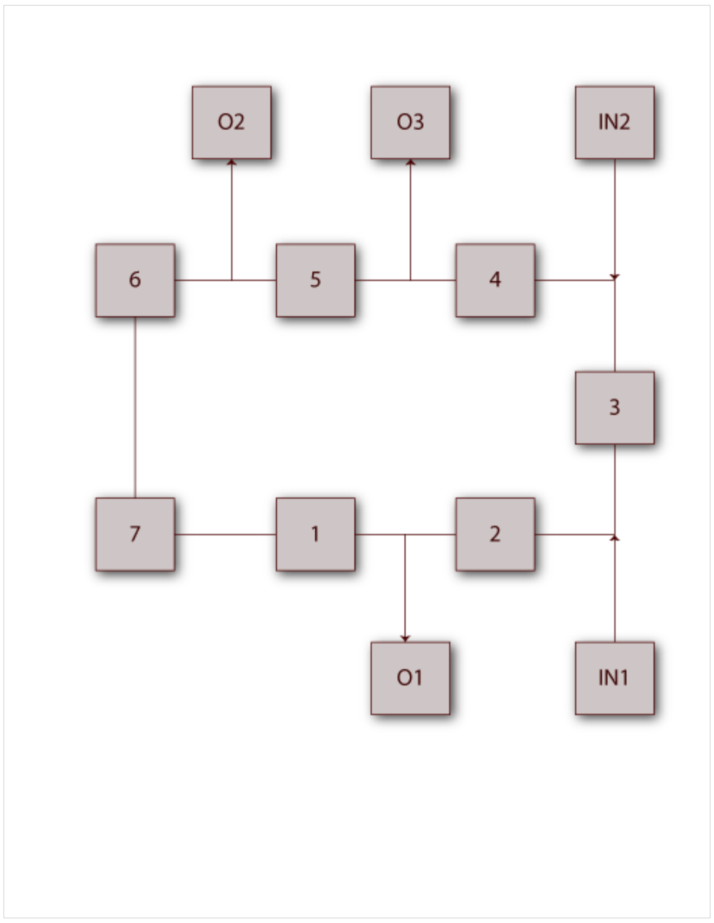


In this diagram $Bm$is a mirrored block of B. Since $B$and $Bm$are identical, this diagram is equivalent to a diagram with A, $B$and Cin series, as shown next:

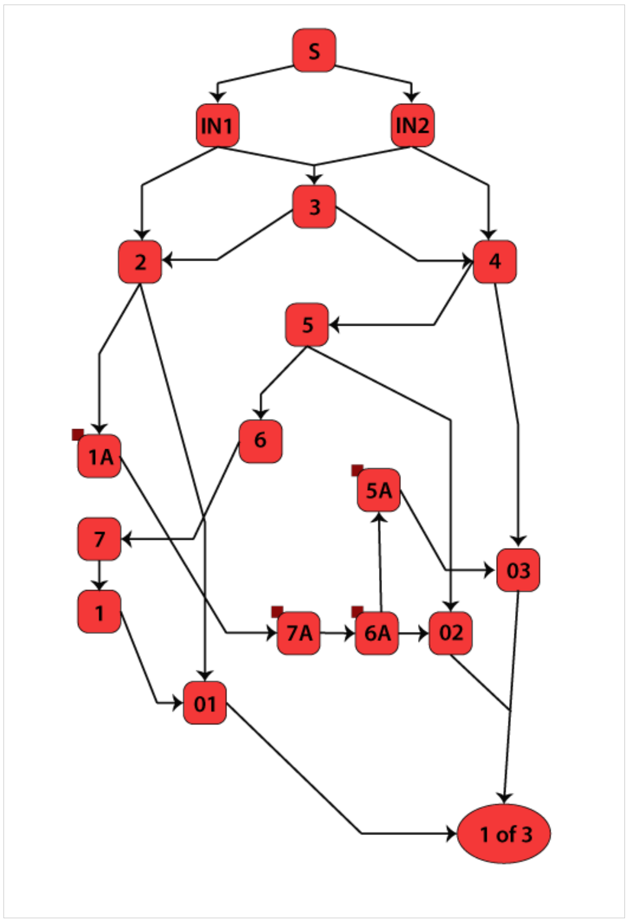


Example: Using Mirrored Blocks

In the diagram shown below, electricity can flow in both directions. Successful system operation requires at least one output (O1, O2 or O3) to be working. Create a block diagram for this system.



The bidirectionality of this system can be modeled using mirrored blocks. The RBD is shown next, where blocks 5A, 7A and 1A are duplicates (or mirrored blocks) of 5, 7 and 1 respectively.



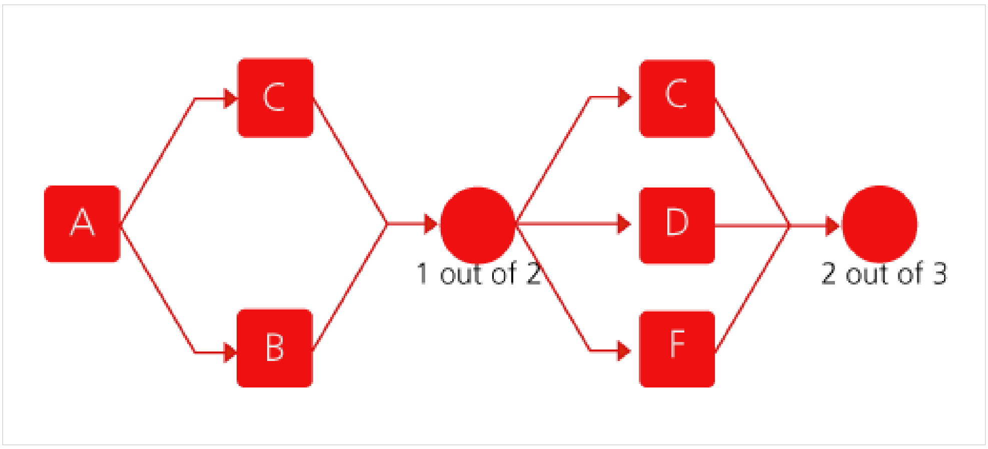
RBDs for Failure Modes and Other Applications

In this reference, most of the examples and derivations assume that each block represents a component/subassembly in a larger system. The same methodology and principles can also be used for other applications. For example, all derivations assume that the event under consideration is the event of failure of a component. One can easily take this principle and apply it to failure modes for a component/subsystem or system. To illustrate this, consider the following examples.

Example: RBDs for Failure Mode Analysis

Assume that a system has five failure modes: A, B, C, D and F. Furthermore, assume that failure of the entire system will occur if mode A occurs, modes B and C occur simultaneously or if either modes C and D, C and F or D and F occur simultaneously. Given the probability of occurrence of each mode, what is the probability of failure of the system?

The following figure shows the diagram for this configuration. Analysis of this diagram follows the same principles as the ones presented in this chapter and can be performed in BlockSim, if desired.

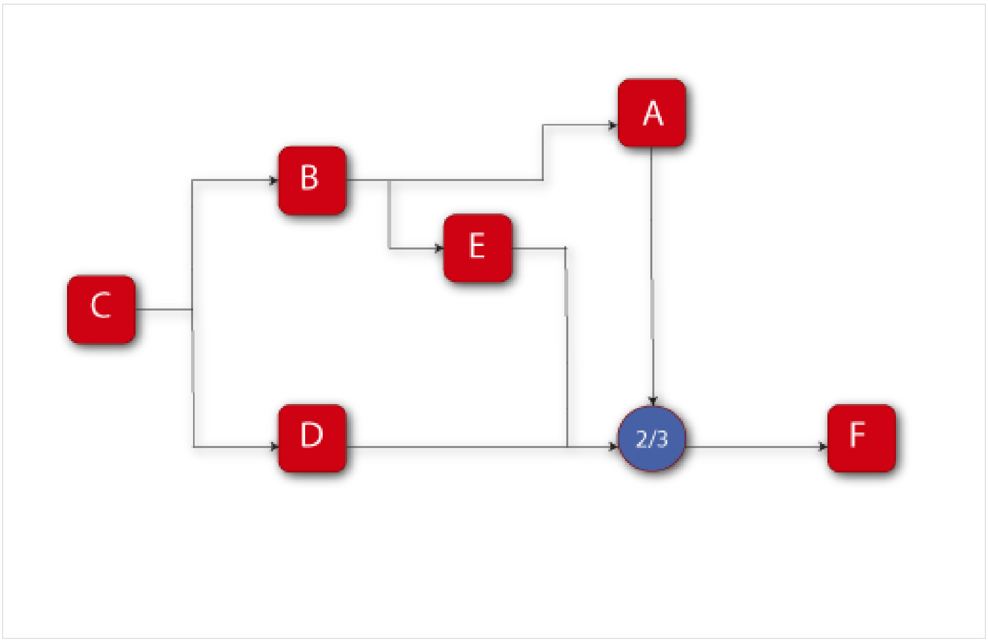


Another Example for Failure Mode Analysis

Assume that a system has six failure modes: A, B, C, D, E and F. Furthermore, assume that failure of the entire system will occur if:

- Mode B, C or F occurs.
- Modes A and E, A and D or E and D occur.

The diagram is shown next:



The reliability equation, as obtained from BlockSim is:

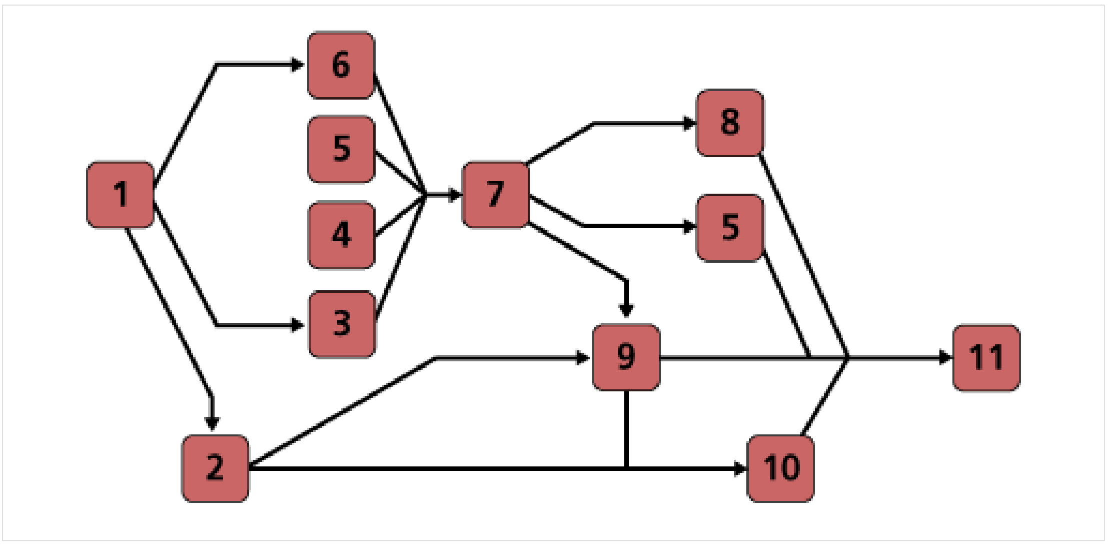
$$\begin{aligned} R_{System} = & (-2R_A \cdot R_B \cdot R_C \cdot R_D \cdot R_{2/3} \cdot R_E \cdot R_F \\ & + R_A \cdot R_B \cdot R_C \cdot R_D \cdot R_{2/3} \cdot R_F \\ & + R_A \cdot R_B \cdot R_C \cdot R_{2/3} \cdot R_E \cdot R_F \\ & + R_B \cdot R_C \cdot R_D \cdot R_{2/3} \cdot R_E \cdot R_F) \end{aligned}$$

The BlockSim equation includes the node reliability term $R_{2/3}$, which cannot fail, or $R_{2/3} = 1$. This can be removed, yielding:

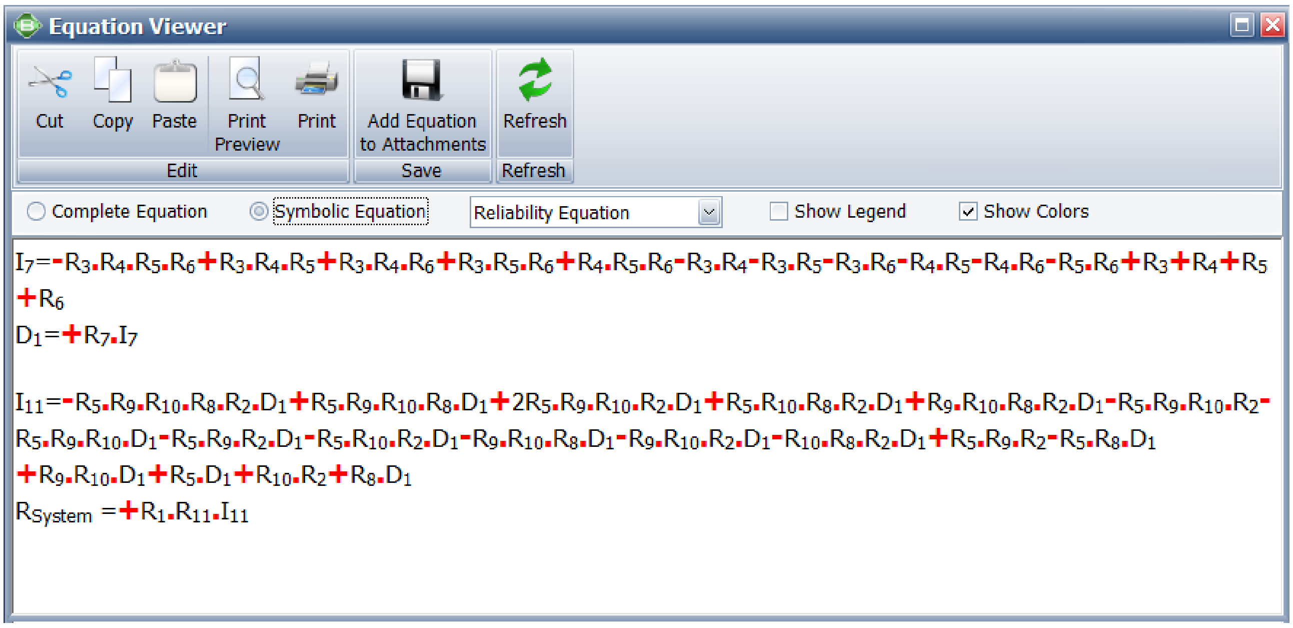
$$\begin{aligned}R_{System} = (&-2R_A \cdot R_B \cdot R_C \cdot R_D \cdot R_E \cdot R_F \\ &+ R_A \cdot R_B \cdot R_C \cdot R_D \cdot R_F \\ &+ R_A \cdot R_B \cdot R_C \cdot R_E \cdot R_F \\ &+ R_B \cdot R_C \cdot R_D \cdot R_E \cdot R_F)\end{aligned}$$

Symbolic Solutions in BlockSim

Several algebraic solutions in BlockSim were used in the prior examples. BlockSim constructs and displays these equations in different ways, depending on the options chosen. As an example, consider the complex system shown next.



When BlockSim constructs the equation internally, it does so in what we will call a symbolic mode. In this mode, portions of the system are segmented. For this example, the symbolic (internal) solution is shown next and composed of the terms shown in the following equations.



Complete Equation [✓ Symbolic Equation] Reliability Equation Show Legend [✓ Show Colors]

$I_7 = -R_3 \cdot R_4 \cdot R_5 \cdot R_6 + R_3 \cdot R_4 \cdot R_5 + R_3 \cdot R_4 \cdot R_6 + R_3 \cdot R_5 \cdot R_6 + R_4 \cdot R_5 \cdot R_6 - R_3 \cdot R_4 - R_3 \cdot R_5 - R_3 \cdot R_6 - R_4 \cdot R_5 - R_4 \cdot R_6 - R_5 \cdot R_6 + R_3 + R_4 + R_5 + R_6$

$D_1 = +R_7 \cdot I_7$

$I_{11} = -R_5 \cdot R_9 \cdot R_{10} \cdot R_8 \cdot R_2 \cdot D_1 + R_5 \cdot R_9 \cdot R_{10} \cdot R_8 \cdot D_1 + 2R_5 \cdot R_9 \cdot R_{10} \cdot R_2 \cdot D_1 + R_5 \cdot R_{10} \cdot R_8 \cdot R_2 \cdot D_1 + R_9 \cdot R_{10} \cdot R_8 \cdot R_2 \cdot D_1 - R_5 \cdot R_9 \cdot R_{10} \cdot R_2 - R_5 \cdot R_9 \cdot R_{10} \cdot D_1 - R_5 \cdot R_9 \cdot R_2 \cdot D_1 - R_5 \cdot R_{10} \cdot R_2 \cdot D_1 - R_9 \cdot R_{10} \cdot R_8 \cdot D_1 - R_9 \cdot R_{10} \cdot R_2 \cdot D_1 - R_{10} \cdot R_8 \cdot R_2 \cdot D_1 + R_5 \cdot R_9 \cdot R_2 - R_5 \cdot R_8 \cdot D_1 + R_9 \cdot R_{10} \cdot D_1 + R_5 \cdot D_1 + R_{10} \cdot R_2 + R_8 \cdot D_1$

$R_{System} = +R_1 \cdot R_{11} \cdot I_{11}$

$$\begin{aligned}
I_7 = & - R_3 \cdot R_5 \cdot R_6 \cdot R_4 + R_3 \cdot R_5 \cdot R_6 + R_3 \cdot R_5 \cdot R_4 \\
& + R_3 \cdot R_6 \cdot R_4 + R_5 \cdot R_6 \cdot R_4 - R_3 \cdot R_5 - R_3 \cdot R_6 \\
& - R_3 \cdot R_4 - R_5 \cdot R_6 - R_5 \cdot R_4 - R_6 \cdot R_4 + R_3 \\
& + R_5 + R_6 + R_4 \\
D_1 = & +R_7 \cdot I_7 \\
I_{11} = & - R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot D_1 + R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot D_1 \\
& + R_2 \cdot R_9 \cdot R_{10} \cdot R_8 \cdot D_1 + R_2 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot D_1 \\
& - R_2 \cdot R_5 \cdot R_{10} \cdot D_1 - R_2 \cdot R_{10} \cdot R_8 \cdot D_1 + R_9 \cdot R_5 \cdot R_8 \cdot D_1 \\
& - R_2 \cdot R_9 \cdot R_{10} - R_2 \cdot R_9 \cdot D_1 - R_9 \cdot R_5 \cdot D_1 - R_9 \cdot R_8 \cdot D_1 \\
& - R_5 \cdot R_8 \cdot D_1 + R_2 \cdot R_9 + R_2 \cdot R_{10} + R_9 \cdot D_1 \\
& + R_5 \cdot D_1 + R_8 \cdot D_1 \\
R_{System} = & +R_1 \cdot R_{11} \cdot I_{11}
\end{aligned}$$

These terms use tokens to represent portions of the equation. In the symbolic equation setting, one reads the solution from the bottom up, replacing any occurrences of a particular token with its definition. In this case then, and to obtain a system solution, one begins with R_{System}. R_{System}contains the token I_{11}. This is then substituted into R_{System}yielding:

$$\begin{aligned}
R_{System} = & + R_1 \cdot R_{11}(-R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot D_1 \\
& + R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot D_1 + R_2 \cdot R_9 \cdot R_{10} \cdot R_8 \cdot D_1 \\
& + R_2 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot D_1 \\
& - R_2 \cdot R_5 \cdot R_{10} \cdot D_1 - R_2 \cdot R_{10} \cdot R_8 \cdot D_1 \\
& + R_9 \cdot R_5 \cdot R_8 \cdot D_1 - R_2 \cdot R_9 \cdot R_{10} \\
& - R_2 \cdot R_9 \cdot D_1 - R_9 \cdot R_5 \cdot D_1 - R_9 \cdot R_8 \cdot D_1 \\
& - R_5 \cdot R_8 \cdot D_1 + R_2 \cdot R_9 \\
& + R_2 \cdot R_{10} + R_9 \cdot D_1 + R_5 \cdot D_1 + R_8 \cdot D_1)
\end{aligned}$$

Now the equation above contains the token D_1. The next step is to substitute D_1into the equation . Doing so yields I_7, shown next.

$$\begin{aligned}
R_{System} = & + R_1 \cdot R_{11}(-R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot (R_7 \cdot I_7) \\
& + R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot (R_7 \cdot I_7) \\
& + R_2 \cdot R_9 \cdot R_{10} \cdot R_8 \cdot (R_7 \cdot I_7) \\
& + R_2 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot (R_7 \cdot I_7) \\
& - R_2 \cdot R_5 \cdot R_{10} \cdot (R_7 \cdot I_7) \\
& - R_2 \cdot R_{10} \cdot R_8 \cdot (R_7 \cdot I_7) + R_9 \cdot R_5 \cdot R_8 \cdot (R_7 \cdot I_7) \\
& - R_2 \cdot R_9 \cdot R_{10} \\
& - R_2 \cdot R_9 \cdot (R_7 \cdot I_7) - R_9 \cdot R_5 \cdot (R_7 \cdot I_7) \\
& - R_9 \cdot R_8 \cdot (R_7 \cdot I_7) - R_5 \cdot R_8 \cdot (R_7 \cdot I_7) + R_2 \cdot R_9 \\
& + R_2 \cdot R_{10} + R_9 \cdot (R_7 \cdot I_7) \\
& + R_5 \cdot (R_7 \cdot I_7) + R_8 \cdot (R_7 \cdot I_7))
\end{aligned}$$

The equation above contains the token D_1. The last step is then to substitute R_{System}into equation above:

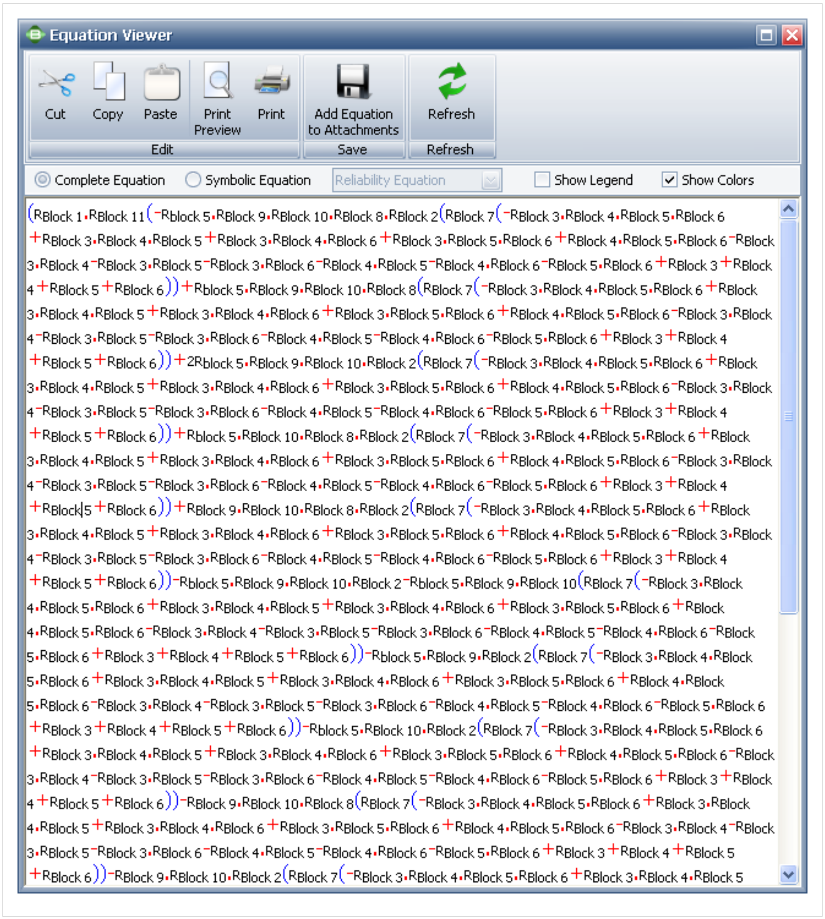
$$R_{System} = +R_1 \cdot R_{11}(-R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot (R_7 \cdot (-R_3 \cdot R_5 \cdot R_6 \cdot R_4 + R_3 \cdot R_5 \cdot R_6 + R_3 \cdot R_5 \cdot R_4 + R_3 \cdot R_6 \cdot R_4 + R_5 \cdot R_6 \cdot R_4 - R_3 \cdot R_5 - R_3 \cdot R_6 - R_3 \cdot R_4 - R_5 \cdot R_6 - R_5 \cdot R_4 - R_6 \cdot R_4 + R_3 + R_5 + R_6 + R_4)) + R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot ($$

When the complete equation is chosen, BlockSim's System Reliability Equation window performs these token substitutions automatically. It should be pointed out that the complete equation can get very large. While BlockSim internally can deal with millions of terms in an equation, the System Reliability Equation window will only format and display equations up to 64,000 characters. BlockSim uses a 64K memory buffer for displaying equations.

Identical Block Simplification Option ("Use IBS")

When computing the system equation with the **Use IBS** option selected, BlockSim looks for identical blocks (blocks with the same failure characteristics) and attempts to simplify the equation, if possible. The symbolic solution for the system in the prior case, with the **Use IBS** option selected and setting equal reliability block properties, is:

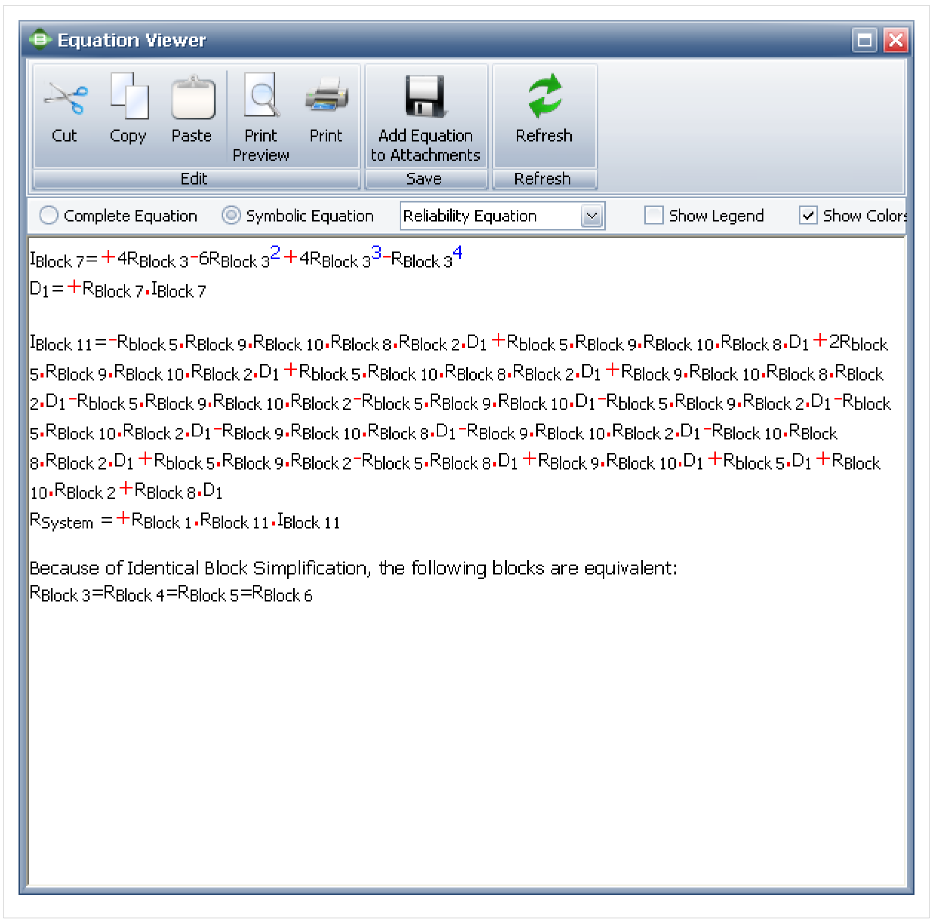
$$
\begin{aligned}
&R_3 = R_6 = R_4 = R_5 \\
&I_7 = +4R_3 - 6R_3^2 + 4R_3^3 - R_3^4 \\
&D_1 = +R_7 \cdot I_7 \\
&I_{11} = -R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot D_1 + R_2 \cdot R_9 \cdot R_5 \cdot R_{10} \cdot D_1 \\
&\qquad + R_2 \cdot R_9 \cdot R_{10} \cdot R_8 \cdot D_1 + R_2 \cdot R_5 \cdot R_{10} \cdot R_8 \cdot D_1 \\
&\qquad - R_2 \cdot R_5 \cdot R_{10} \cdot D_1 - R_2 \cdot R_{10} \cdot R_8 \cdot D_1 + R_9 \cdot R_5 \cdot R_8 \cdot D_1 \\
&\qquad - R_2 \cdot R_9 \cdot R_{10} - R_2 \cdot R_9 \cdot D_1 - R_9 \cdot R_5 \cdot D_1 \\
&\qquad - R_9 \cdot R_8 \cdot D_1 - R_5 \cdot R_8 \cdot D_1 + R_2 \cdot R_9 + R_2 \cdot R_{10} \\
&\qquad + R_9 \cdot D_1 + R_5 \cdot D_1 + R_8 \cdot D_1 \\
&R_{System} = +R_1 \cdot R_{11} \cdot I_{11}
\end{aligned}
$$



[✓ Complete Equation] Symbolic Equation   Show Legend [✓ Show Colors]

```
(RBlock 1·RBlock 11(−Rblock 5·RBlock 9·RBlock 10·RBlock 8·RBlock 2(RBlock 7(−RBlock 3·RBlock 4·RBlock 5·RBlock 6
+RBlock 3·RBlock 4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock 4·RBlock 5·RBlock 6−RBlock
3·RBlock 4−RBlock 3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock 5·RBlock 6+RBlock 3+RBlock
4+RBlock 5+RBlock 6))+Rblock 5·RBlock 9·RBlock 10·RBlock 8(RBlock 7(−RBlock 3·RBlock 4·RBlock 5·RBlock 6+RBlock
3·RBlock 4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock 4·RBlock 5·RBlock 6−RBlock 3·RBlock
4−RBlock 3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock 5·RBlock 6+RBlock 3+RBlock 4
+RBlock 5+RBlock 6))+2Rblock 5·RBlock 9·RBlock 10·RBlock 2(RBlock 7(−RBlock 3·RBlock 4·RBlock 5·RBlock 6+RBlock
3·RBlock 4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock 4·RBlock 5·RBlock 6−RBlock 3·RBlock
4−RBlock 3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock 5·RBlock 6+RBlock 3+RBlock 4
+RBlock 5+RBlock 6))+Rblock 5·RBlock 10·RBlock 8·RBlock 2(RBlock 7(−RBlock 3·RBlock 4·RBlock 5·RBlock 6+RBlock
3·RBlock 4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock 4·RBlock 5·RBlock 6−RBlock 3·RBlock
4−RBlock 3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock 5·RBlock 6+RBlock 3+RBlock 4
+RBlock 5+RBlock 6))+RBlock 9·RBlock 10·RBlock 8·RBlock 2(RBlock 7(−RBlock 3·RBlock 4·RBlock 5·RBlock 6+RBlock
3·RBlock 4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock 4·RBlock 5·RBlock 6−RBlock 3·RBlock
4−RBlock 3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock 5·RBlock 6+RBlock 3+RBlock 4
+RBlock 5+RBlock 6))−Rblock 5·RBlock 9·RBlock 10·RBlock 2−Rblock 5·RBlock 9·RBlock 10(RBlock 7(−RBlock 3·RBlock
4·RBlock 5·RBlock 6+RBlock 3·RBlock 4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock
4·RBlock 5·RBlock 6−RBlock 3·RBlock 4−RBlock 3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock
5·RBlock 6+RBlock 3+RBlock 4+RBlock 5+RBlock 6))−Rblock 5·RBlock 9·RBlock 2(RBlock 7(−RBlock 3·RBlock 4·RBlock
5·RBlock 6+RBlock 3·RBlock 4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock 4·RBlock
5·RBlock 6−RBlock 3·RBlock 4−RBlock 3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock 5·RBlock 6
+RBlock 3+RBlock 4+RBlock 5+RBlock 6))−Rblock 5·RBlock 10·RBlock 2(RBlock 7(−RBlock 3·RBlock 4·RBlock 5·RBlock 6
+RBlock 3·RBlock 4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock 4·RBlock 5·RBlock 6−RBlock
3·RBlock 4−RBlock 3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock 5·RBlock 6+RBlock 3+RBlock
4+RBlock 5+RBlock 6))−RBlock 9·RBlock 10·RBlock 8(RBlock 7(−RBlock 3·RBlock 4·RBlock 5·RBlock 6+RBlock 3·RBlock
4·RBlock 5+RBlock 3·RBlock 4·RBlock 6+RBlock 3·RBlock 5·RBlock 6+RBlock 4·RBlock 5·RBlock 6−RBlock 3·RBlock 4−RBlock
3·RBlock 5−RBlock 3·RBlock 6−RBlock 4·RBlock 5−RBlock 4·RBlock 6−RBlock 5·RBlock 6+RBlock 3+RBlock 4+RBlock 5
+RBlock 6))−RBlock 9·RBlock 10·RBlock 2(RBlock 7(−RBlock 3·RBlock 4·RBlock 5·RBlock 6+RBlock 3·RBlock 4·RBlock 5
```

When using IBS, the resulting equation is invalidated if any of the block properties (e.g., failure distributions) have changed since the equation was simplified based on those properties.



Complete Equation [✓ Symbolic Equation] Reliability Equation ☐ Show Legend ☑ Show Colors

$I_{Block\ 7} = +4R_{Block\ 3} - 6R_{Block\ 3}^{2} + 4R_{Block\ 3}^{3} - R_{Block\ 3}^{4}$

$D_1 = +R_{Block\ 7} \cdot I_{Block\ 7}$

$$\begin{aligned} I_{Block\ 11} = {} & -R_{block\ 5} \cdot R_{Block\ 9} \cdot R_{Block\ 10} \cdot R_{Block\ 8} \cdot R_{Block\ 2} \cdot D_1 + R_{block\ 5} \cdot R_{Block\ 9} \cdot R_{Block\ 10} \cdot R_{Block\ 8} \cdot D_1 + 2R_{block\ 5} \cdot R_{Block\ 9} \cdot R_{Block\ 10} \cdot R_{Block\ 2} \cdot D_1 \\ & + R_{block\ 5} \cdot R_{Block\ 10} \cdot R_{Block\ 8} \cdot R_{Block\ 2} \cdot D_1 + R_{Block\ 9} \cdot R_{Block\ 10} \cdot R_{Block\ 8} \cdot R_{Block\ 2} \cdot D_1 - R_{block\ 5} \cdot R_{Block\ 9} \cdot R_{Block\ 10} \cdot R_{Block\ 2} \\ & - R_{block\ 5} \cdot R_{Block\ 9} \cdot R_{Block\ 10} \cdot D_1 - R_{block\ 5} \cdot R_{Block\ 9} \cdot R_{Block\ 2} \cdot D_1 - R_{block\ 5} \cdot R_{Block\ 10} \cdot R_{Block\ 2} \cdot D_1 - R_{Block\ 9} \cdot R_{Block\ 10} \cdot R_{Block\ 8} \cdot D_1 \\ & - R_{Block\ 9} \cdot R_{Block\ 10} \cdot R_{Block\ 2} \cdot D_1 - R_{Block\ 10} \cdot R_{Block\ 8} \cdot R_{Block\ 2} \cdot D_1 + R_{block\ 5} \cdot R_{Block\ 9} \cdot R_{Block\ 2} - R_{block\ 5} \cdot R_{Block\ 8} \cdot D_1 \\ & + R_{Block\ 9} \cdot R_{Block\ 10} \cdot D_1 + R_{block\ 5} \cdot D_1 + R_{Block\ 10} \cdot R_{Block\ 2} + R_{Block\ 8} \cdot D_1 \end{aligned}$$

$R_{System} = +R_{Block\ 1} \cdot R_{Block\ 11} \cdot I_{Block\ 11}$

Because of Identical Block Simplification, the following blocks are equivalent:
$R_{Block\ 3} = R_{Block\ 4} = R_{Block\ 5} = R_{Block\ 6}$

Chapter 4

Time-Dependent System Reliability (Analytical)

In the RBDs and Analytical System Reliability chapter, different system configuration types were examined, as well as different methods for obtaining the system's reliability function analytically. Because the reliabilities in the problems presented were treated as probabilities (e.g., $P(A)$, R_i), the reliability values and equations presented were referred to as *static* (not time-dependent). Thus, in the prior chapter, the life distributions of the components were not incorporated in the process of calculating the system reliability. In this chapter, time dependency in the reliability function will be introduced. We will develop the models necessary to observe the reliability over the life of the system, instead of at just one point in time. In addition, performance measures such as failure rate, MTTF and warranty time will be estimated for the entire system. The methods of obtaining the reliability function analytically remain identical to the ones presented in the previous chapter, with the exception that the reliabilities will be functions of time. In other words, instead of dealing with R_i, we will use $R_i(t)$. All examples in this chapter assume that no repairs are performed on the components. Repairable systems analysis will be introduced in a subsequent chapter.

Analytical Life Predictions

The analytical approach presented in the prior chapter involved the determination of a mathematical expression that describes the reliability of the system, expressed in terms of the reliabilities of its components. So far we have estimated only static system reliability (at a fixed time). For example, in the case of a system with three components in series, the system's reliability equation was given by:

$$R_s = R_1 \cdot R_2 \cdot R_3$$

The values of R_1, $R_2$and R_3were given for a common time and the reliability of the system was estimated for that time. However, since the component failure characteristics can be described by distributions, the system reliability is actually time-dependent. In this case, the equation above can be rewritten as:

$$R_s(t) = R_1(t) \cdot R_2(t) \cdot R_3(t)$$

The reliability of the system for any mission time can now be estimated. Assuming a Weibull life distribution for each component, the first equation above can now be expressed in terms of each component's reliability function, or:

$$R_s(t) = e^{-\left(\frac{t}{\eta_1}\right)^{\beta_1}} \cdot e^{-\left(\frac{t}{\eta_2}\right)^{\beta_2}} \cdot e^{-\left(\frac{t}{\eta_3}\right)^{\beta_3}}$$

In the same manner, any life distribution can be substituted into the system reliability equation. Suppose that the times-to-failure of the first component are described with a Weibull distribution, the times-to-failure of the second component with an exponential distribution and the times-to-failure of the third component with a normal distribution. Then the first equation above can be written as:

$$R_s(t) = e^{-\left(\frac{t}{\eta_1}\right)^{\beta_1}} \cdot e^{-\lambda_2 t} \cdot \left[1 - \Phi\left(\frac{t - \mu_3}{\sigma_3}\right)\right]$$

It can be seen that the biggest challenge is in obtaining the system's reliability function in terms of component reliabilities, which has already been discussed in depth. Once this has been achieved, calculating the reliability of the system for any mission duration is just a matter of substituting the corresponding component reliability functions into the system reliability equation.

Advantages and Disadvantages

The primary advantage of the analytical solution is that it produces a mathematical expression that describes the reliability of the system. Once the system's reliability function has been determined, other calculations can then be performed to obtain metrics of interest for the system. Such calculations include:

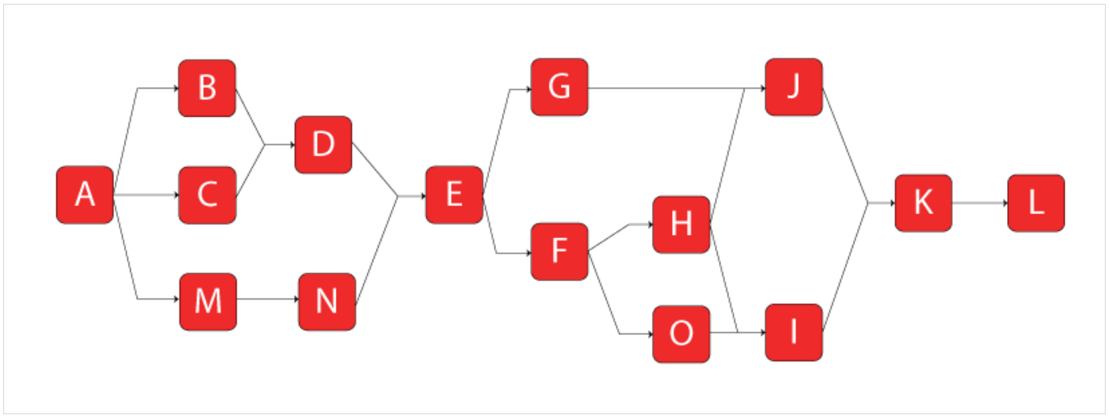
- Determination of the system's *pdf*.
- Determination of warranty periods.
- Determination of the system's failure rate.
- Determination of the system's MTTF.

In addition, optimization and reliability allocation techniques can be used to aid engineers in their design improvement efforts. Another advantage in using analytical techniques is the ability to perform static calculations and analyze systems with a mixture of static and time-dependent components. Finally, the reliability importance of components over time can be calculated with this methodology.

The biggest disadvantage of the analytical method is that formulations can become very complicated. The more complicated a system is, the larger and more difficult it will be to analytically formulate an expression for the system's reliability. For particularly detailed systems this process can be quite time-consuming, even with the use of computers. Furthermore, when the maintainability of the system or some of its components must be taken into consideration, analytical solutions become intractable. In these situations, the use of simulation methods may be more advantageous than attempting to develop a solution analytically. Simulation methods are presented in later chapters.

Looking at a Simple "Complex" System Analytically

The complexity involved in an analytical solution can be best illustrated by looking at the simple *complex* system with 15 components, as shown below.



The system reliability for this system (computed using BlockSim) is shown next. The first solution is provided using BlockSim's symbolic solution. In symbolic mode, BlockSim breaks the equation into segments, identified by tokens, that need to be substituted into the final system equation for a complete solution. This creates algebraic solutions that are more compact than if the substitutions were made.

$$\begin{aligned}
R_{System} =& D2 \cdot D3 \cdot R_L \\
& D3 = +R_K \cdot IK \\
& IK = +R_I \cdot R_J \cdot R_O \cdot R_G \cdot R_F \cdot R_H - R_I \cdot R_J \cdot R_O \cdot R_G \cdot R_F \\
& \qquad - R_I \cdot R_J \cdot R_F \cdot R_H - R_I \cdot R_O \cdot R_F \cdot R_H \\
& \qquad - R_J \cdot R_G \cdot R_F \cdot R_H + R_I \cdot R_O \cdot R_F \\
& \qquad + R_I \cdot R_F \cdot R_H + R_J \cdot R_F \cdot R_H + R_J \cdot R_G \\
& D2 = +R_A \cdot R_E \cdot IE \\
& IE = -D1 \cdot R_M \cdot R_N + R_M \cdot R_N + D1 \\
& D1 = +R_D \cdot ID \\
& ID = -R_B \cdot R_C + R_B + R_C
\end{aligned}$$

Substituting the terms yields:

$$\begin{aligned}
R_{System} =& R_A \cdot R_E \cdot R_L \cdot R_K \\
& \cdot \{(R_D \cdot R_B \cdot R_C + R_B + R_C) \cdot R_M \cdot R_N \\
& + R_M \cdot R_N - R_D \cdot R_B \cdot R_C + R_B + R_C\} \\
& \cdot \{R_I \cdot R_J \cdot R_O \cdot R_G \cdot R_F \cdot R_H - R_I \cdot R_J \cdot R_O \cdot R_G \cdot R_F \\
& - R_I \cdot R_J \cdot R_F \cdot R_H - R_I \cdot R_O \cdot R_F \cdot R_H \\
& - R_J \cdot R_G \cdot R_F \cdot R_H + R_I \cdot R_O \cdot R_F \\
& + R_I \cdot R_F \cdot R_H + R_J \cdot R_F \cdot R_H + R_J \cdot R_G\}
\end{aligned}$$

BlockSim's automatic algebraic simplification would yield the following format for the above solution:

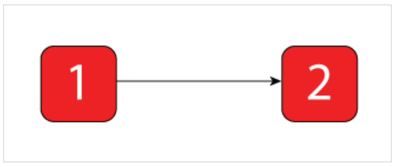
$$\begin{aligned}
R_{System} =& ((R_A \cdot R_E(-(R_D(-R_B \cdot R_C + R_B + R_C))R_M \cdot R_N \\
& + R_M \cdot R_N \\
& + (R_D(-R_B \cdot R_C + R_B + R_C)))) \\
& (R_K(R_I \cdot R_J \cdot R_O \cdot R_G \cdot R_F \cdot R_H \\
& - R_I \cdot R_J \cdot R_O \cdot R_G \cdot R_F - R_I \cdot R_J \cdot R_F \cdot R_H \\
& - R_I \cdot R_O \cdot R_F \cdot R_H - R_J \cdot R_G \cdot R_F \cdot R_H \\
& + RI \cdot R_O \cdot R_F \\
& + R_I \cdot R_F \cdot R_H + R_J \cdot R_F \cdot R_H + R_J \cdot R_G))R_L)
\end{aligned}$$

In this equation, each $R_i$represents the reliability function of a block. For example, if $R_A$has a Weibull distribution, then each $R_A(t) = e^{-\left(\frac{t}{\eta_A}\right)^{\beta_A}}$ and so forth. Substitution of each component's reliability function in the last R_{System}equation above will result in an analytical expression for the system reliability as a function of time, or $R_s(t)$, which is the same as $(1 - cdf_{System})$.

Obtaining Other Functions of Interest

Once the system reliability equation (or the cumulative density function, *cdf*) has been determined, other functions and metrics of interest can be derived.

Consider the following simple system:



Furthermore, assume that component 1 follows an exponential distribution with a mean of 10,000 ($\mu = 10,000$, $\lambda = 1/10,000$)and component 2 follows a Weibull distribution with $\beta = 6$and $\eta = 10,000$. The reliability equation of this system is:

$$\begin{aligned} R_S(t) =& R_1(t) \cdot R_2(t) \\ =& e^{-\lambda t} \cdot e^{-\left(\frac{t}{\eta}\right)^{\beta}} \\ =& e^{-\frac{1}{10{,}000}t} \cdot e^{-\left(\frac{t}{10{,}000}\right)^{6}} \end{aligned}$$

The system *cdf* is:

$$\begin{aligned} F_S(t) =& 1 - (R_1(t) \cdot R_2(t)) \\ =& 1 - \left(e^{-\lambda t} \cdot e^{-\left(\frac{t}{\eta}\right)^{\beta}}\right) \\ =& 1 - \left(e^{-\frac{1}{10{,}000}t} \cdot e^{-\left(\frac{t}{10{,}000}\right)^{6}}\right) \end{aligned}$$

System *pdf*

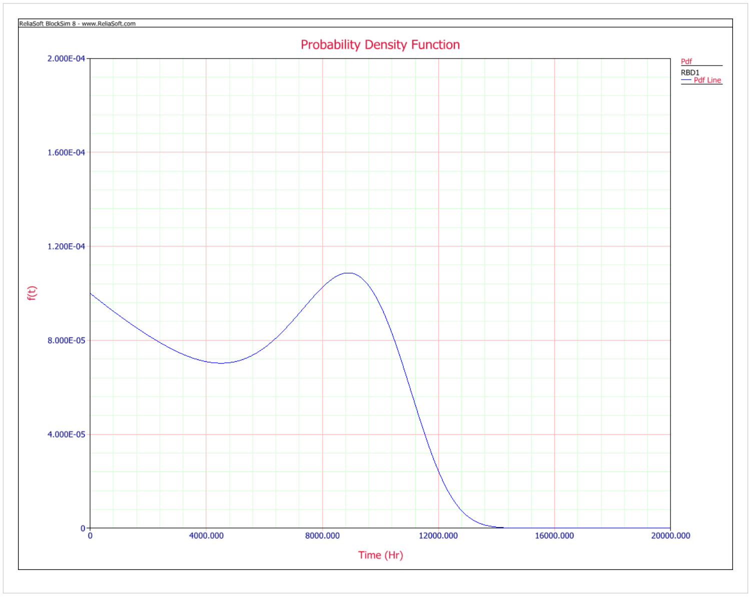
Once the equation for the reliability of the system has been obtained, the system's *pdf* can be determined. The *pdf* is the derivative of the reliability function with respect to time or:

$$f_s(t) = -\frac{d(R_s(t))}{dt}$$

For the system shown above, this is:

$$\begin{aligned} f_s(t) =& -\frac{d}{dt}\left(e^{-\frac{1}{10{,}000}t} \cdot e^{-\left(\frac{t}{10{,}000}\right)^{6}}\right) \\ =& -\frac{d}{dt}\left(e^{-\frac{1}{10{,}000}t}\right) \cdot e^{-\left(\frac{t}{10{,}000}\right)^{6}} \\ & + e^{-\frac{1}{10{,}000}t}\left[-\frac{d}{dt}\left(e^{-\left(\frac{t}{10{,}000}\right)^{6}}\right)\right] \\ =& f_1(t) \cdot R_2(t) + f_2(t) \cdot R_1(t) \end{aligned}$$

The next figure shows a plot of the *pdf* equation.



Conditional Reliability

Conditional reliability is the probability of a system successfully completing another mission following the successful completion of a previous mission. The time of the previous mission and the time for the mission to be undertaken must be taken into account for conditional reliability calculations. The system's conditional reliability function is given by:

$$R(T,t)=\frac{R(T+t)}{R(T)}$$

Equation above gives the reliability for a new mission of duration $t$having already accumulated Thours of operation up to the start of this new mission. The system is evaluated to assure that it will start the next mission successfully.

For the simple two-component system, the reliability for mission of $t=1,000$hours, having an age of $T=500$ hours, is:

$$\begin{aligned}R_S(T=500,t=1000)=&\frac{R(T+t)}{R(T)}\\ =&\frac{R(1500)}{R(500)}\\ =&\frac{e^{-\frac{1500}{10,000}}\cdot e^{-\left(\frac{1500}{10,000}\right)^6}}{e^{-\frac{500}{10,000}t}\cdot e^{-\left(\frac{500}{10,000}\right)^6}}\\ =&0.9048=90.48\%\end{aligned}$$

Conditional Reliability for Components

Now in this formulation, it was assumed that the accumulated age was equivalent for both units. That is, both started life at zero and aged to 500. It is possible to consider an individual component that has already accumulated some age (used component) in the same formulation. To illustrate this, assume that component 2 started life with an age of T = 100. Then the reliability equation of the system, as given in $R_S(t)=R_1(t)\cdot R_2(t)$, would need to be modified to include a conditional term for 2, or:

$$R_S(t)=R_1(t)\cdot\frac{R_2(T_2+t)}{R_2(T_2)}$$

In BlockSim, the start age input box may be used to specify a starting age greater than zero.

System Failure Rate

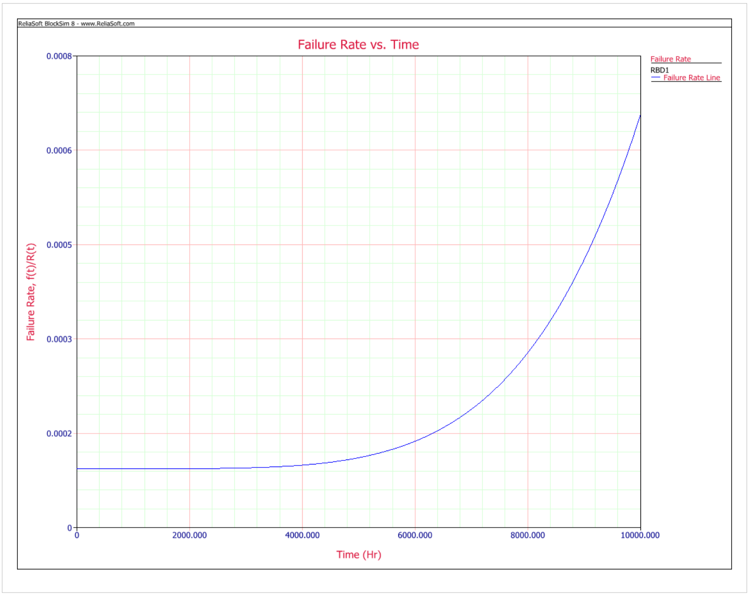
Once the distribution of the system has been determined, the failure rate can also be obtained by dividing the *pdf* by the reliability function:

$$\lambda_s\left(t\right)=\frac{f_s\left(t\right)}{R_s\left(t\right)}$$

For the simple two-component system:

$$\begin{aligned}
\lambda_s(t) = & \frac{-\frac{d}{dt}\left(e^{-\frac{1}{10{,}000}t}\cdot e^{-\left(\frac{t}{10{,}000}\right)^6}\right)}{e^{-\frac{1}{10{,}000}t}\cdot e^{-\left(\frac{t}{10{,}000}\right)^6}} \\
= & \frac{-\frac{d}{dt}\left(e^{-\frac{1}{10{,}000}t}\right)}{e^{-\frac{1}{10{,}000}t}} + \frac{-\frac{d}{dt}\left(e^{-\left(\frac{t}{10{,}000}\right)^6}\right)}{e^{-\left(\frac{t}{10{,}000}\right)^6}} \\
= & \frac{f_1}{R_1} + \frac{f_2}{R_2} \\
= & \lambda_1 + \lambda_2
\end{aligned}$$

The following figure shows a plot of the equation.



BlockSim uses numerical methods to estimate the failure rate. It should be pointed out that as $t \rightarrow \infty$, numerical evaluation of the first equation above is constrained by machine numerical precision. That is, there are limits as to how large $t$can get before floating point problems arise. For example, at $t = 5,000,000$both numerator and denominator will tend to zero (e.g., $e^{-\frac{5{,}000{,}000}{10{,}000}} = 7.1245 \times 10^{-218}$). As these numbers become very small they will start looking like a zero to the computer, or cause a floating point error, resulting in a $\frac{0}{0}$or $\frac{X}{0}$operation. In these cases, BlockSim will return a value of " N/A" for the result. Obviously, this does not create any practical constraints.

System Mean Life (Mean Time To Failure)

The mean life (or mean time to failure, MTTF) can be obtained by integrating the system reliability function from zero to infinity:

$$MTTF = \int_0^\infty R_s(t)\,dt$$

The mean time is a performance index and does not provide any information about the behavior of the failure distribution of the system.

For the simple two-component system:

$$\begin{aligned}
MTTF = & \int_0^\infty \left(e^{-\frac{1}{10{,}000}t}\cdot e^{-\left(\frac{t}{10{,}000}\right)^6}\right) dt \\
= & 5978.9
\end{aligned}$$

Warranty Period and BX Life

Sometimes it is desirable to know the time value associated with a certain reliability. Warranty periods are often calculated by determining what percentage of the failure population can be covered financially and estimating the time at which this portion of the population will fail. Similarly, engineering specifications may call for a certain BX life, which also represents a time period during which a certain proportion of the population will fail. For example, the B10 life is the time in which 10% of the population will fail. This is obtained by setting $R_S(t)$to the desired value and solving for t.

For the simple two-component system:

$$R_s\left(t\right)=e^{-\frac{1}{10{,}000}t}\cdot e^{-\left(\frac{t}{10{,}000}\right)^6}$$

To compute the time by which reliability would be equal to 90%, equation above is recast as follows and solved for t.

$$0.90=e^{-\frac{1}{10{,}000}t}\cdot e^{-\left(\frac{t}{10{,}000}\right)^6}$$

In this case, $t=1053.59$. Equivalently, the B10 life for this system is also 1053.59. Except for some trivial cases, a closed form solution for $t$cannot be obtained. Thus, it is necessary to solve for $t$using numerical methods. BlockSim uses numerical methods.

Examples

Components in Series

Consider a system consisting of 3 exponential units, connected in series, with the following failure rates (in failures per hour): $\lambda_1=0.0002$, $\lambda_2=0.0005$and $\lambda_3=0.0001$.

- Obtain the reliability equation for the system.
- What is the reliability of the system after 150 hours of operation?
- Obtain the system's *pdf*.
- Obtain the system's failure rate equation.
- What is the MTTF for the system?
- What should the warranty period be for a 90% reliability?

Solution

The analytical expression for the reliability of the system is given by:

$$\begin{aligned}R_s(t)=&R_1(t)\cdot R_2(t)\cdot R_3(t)\\=&e^{-\lambda_1 t}\cdot e^{-\lambda_2 t}\cdot e^{-\lambda_1 t}\\=&e^{-(\lambda_1+\lambda_2+\lambda_3)t}\end{aligned}$$

At 150 hours of operation, the reliability of the system is:

$$\begin{aligned}R_s\left(t\right)=&e^{-(0.0002+0.0005+0.0001)150}\\=&0.8869\text{ or }88.69\%\end{aligned}$$

In order to obtain the system's *pdf*, the derivative of the reliability equation given in the first equation above is taken with respect to time, or:

$$\begin{aligned}f_s(t)=&-\frac{d[R_s(t)]}{dt}\\=&-\frac{d\left[e^{-(\lambda_1+\lambda_2+\lambda_3)t}\right]}{dt}\\=&(\lambda_1+\lambda_2+\lambda_3)\cdot e^{-(\lambda_1+\lambda_2+\lambda_3)t}\end{aligned}$$

The system's failure rate can now be obtained simply by dividing the system's *pdf* given in the equation above by the system's reliability function given in the first equation above, and:

$$\begin{aligned}\lambda_s(t) =& \frac{f_s(t)}{R_s(t)}\\ =& \frac{(\lambda_1+\lambda_2+\lambda_3)\cdot e^{-(\lambda_1+\lambda_2+\lambda_3)t}}{e^{-(\lambda_1+\lambda_2+\lambda_3)t}}\\ =&(\lambda_1+\lambda_2+\lambda_3)\\ =&0.0008\ fr/hr\end{aligned}$$

Combining $MTTF = \int_0^\infty R_s(t)\,dt$and the first equation above, the system's MTTF can be obtained:

$$\begin{aligned}MTTF =& \int_0^\infty R_s(t)\,dt\\ =& \int_0^\infty e^{-(\lambda_1+\lambda_2+\lambda_3)t}dt\\ =&\frac{1}{(\lambda_1+\lambda_2+\lambda_3)}\\ =&1250\ hr\end{aligned}$$

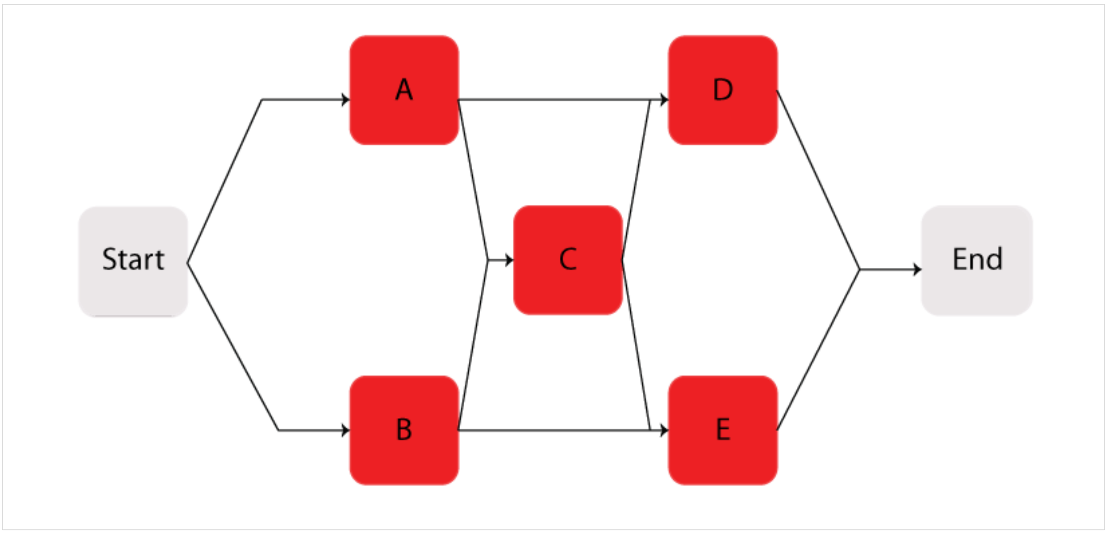
Solving the first equation above with respect to time will yield the corresponding warranty period for a 90% reliability. In this case, the system reliability equation is simple and a closed form solution exists. The warranty time can now be found by solving:

$$\begin{aligned}t =& -\frac{\ln(R)}{\lambda_1+\lambda_2+\lambda_3}\\ =& -\frac{\ln(0.9)}{0.0008}\\ =&131.7\ hr\end{aligned}$$

Thus, the warranty period should be 132 hours.

Components in Parallel

Consider the system shown next.



Components Athrough $E$are Weibull distributed with $\beta = 1.2$and $\eta = 1230$hours. The starting and ending blocks cannot fail.

Determine the following:

- The reliability equation for the system and its corresponding plot.
- The system's *pdf* and its corresponding plot.
- The system's failure rate equation and the corresponding plot.
- The MTTF.
- The warranty time for a 90% reliability.
- The reliability for a 200-hour mission, if it is known that the system has already successfully operated for 200 hours.

Solution

The first step is to obtain the reliability function for the system. The methods described in the RBDs and Analytical System Reliability chapter can be employed, such as the event space or path-tracing methods. Using BlockSim, the following reliability equation is obtained:

$$\begin{aligned} R_s(t) = & (R_{Start}\cdot R_{End}(2R_A\cdot R_D\cdot R_C\cdot R_B\cdot R_E \\ & -R_A\cdot R_D\cdot R_C\cdot R_B-R_A\cdot R_D\cdot R_C\cdot R_E \\ & -R_A\cdot R_D\cdot R_B\cdot R_E-R_A\cdot R_C\cdot R_B\cdot R_E \\ & -R_D\cdot R_C\cdot R_B\cdot R_E+R_A\cdot R_C\cdot R_E \\ & +R_D\cdot R_C\cdot R_B+R_A\cdot R_D+R_B\cdot R_E))\end{aligned}$$

Note that since the starting and ending blocks cannot fail, $R_{Start}=1$and $R_{End}=1$,the equation above can be reduced to:

$$\begin{aligned} R_s(t) = & 2\cdot R_A\cdot R_D\cdot R_C\cdot R_B\cdot R_E \\ & -R_A\cdot R_D\cdot R_C\cdot R_B-R_A\cdot R_D\cdot R_C\cdot R_E \\ & -R_A\cdot R_D\cdot R_B\cdot R_E-R_A\cdot R_C\cdot R_B\cdot R_E \\ & -R_D\cdot R_C\cdot R_B\cdot R_E+R_A\cdot R_C\cdot R_E \\ & +R_D\cdot R_C\cdot R_B+R_A\cdot R_D+R_B\cdot R_E\end{aligned}$$

where $R_A$is the reliability equation for Component A, or:

$$R_A(t)=e^{-\left(\frac{t}{\eta_A}\right)^{\beta_A}}$$

$R_B$is the reliability equation for Component B, etc.

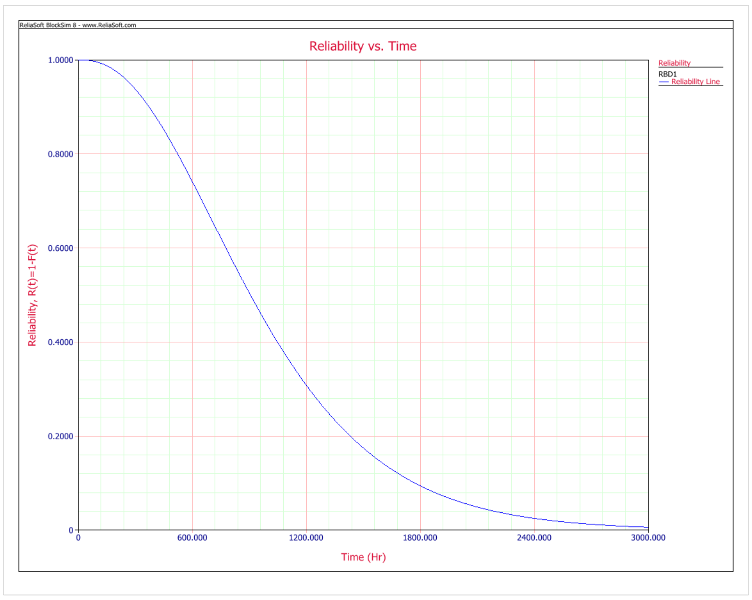
Since the components in this example are identical, the system reliability equation can be further reduced to:

$$R_s(t)=2R(t)^2+2R(t)^3-5R(t)^4+2R(t)^5$$

Or, in terms of the failure distribution:

$$R_s(t)=2\cdot e^{-2\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-3\left(\frac{t}{\eta}\right)^{\beta}}-5\cdot e^{-4\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-5\left(\frac{t}{\eta}\right)^{\beta}}$$

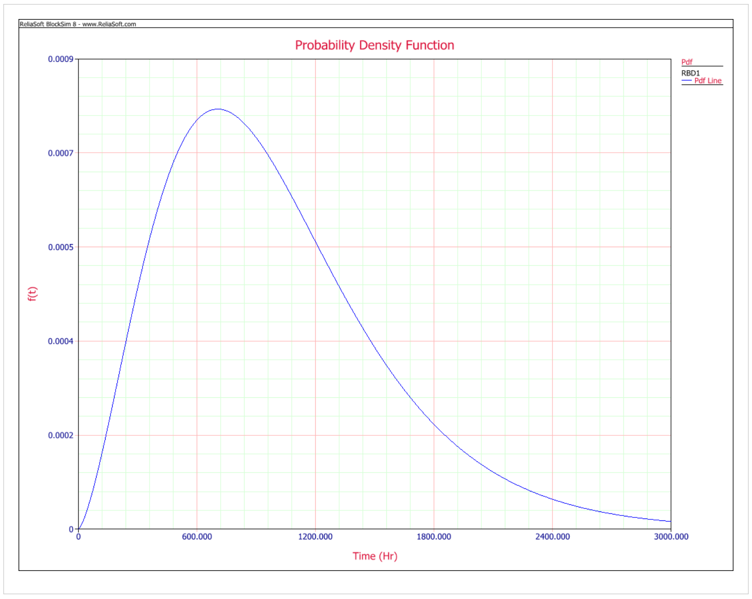
The corresponding plot is given in the following figure.



In order to obtain the system's *pdf*, the derivative of the reliability equation given above is taken with respect to time, resulting in:

$$f_s(t) = 4\cdot\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}e^{-2\left(\frac{t}{\eta}\right)^{\beta}} + 6\cdot\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}e^{-3\left(\frac{t}{\eta}\right)^{\beta}} - 20\cdot\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}e^{-4\left(\frac{t}{\eta}\right)^{\beta}} + 10\cdot\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}e^{-5\left(\frac{t}{\eta}\right)^{\beta}}$$

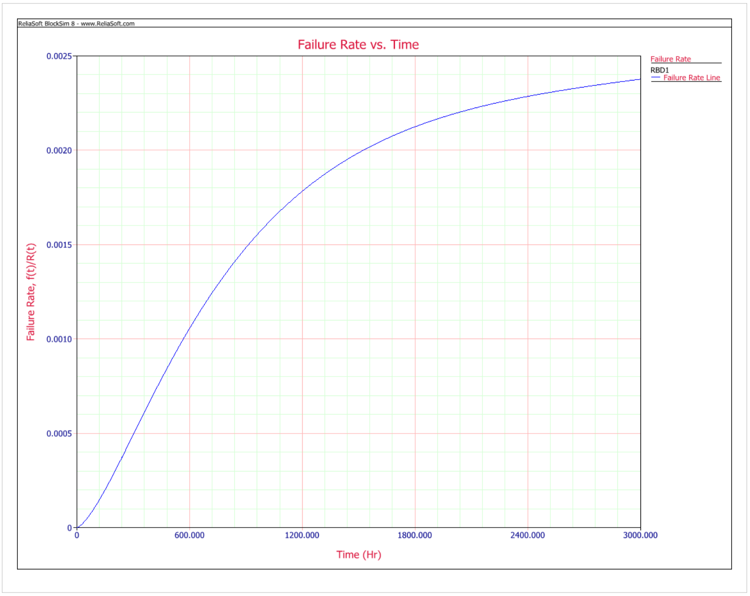
The *pdf* can now be plotted for different time values, t, as shown in the following figure.



The system's failure rate can be obtained by dividing the system's *pdf*, given in equation above, by the system's reliability function given in $R_s(t) = 2\cdot e^{-2\left(\frac{t}{\eta}\right)^{\beta}} + 2\cdot e^{-3\left(\frac{t}{\eta}\right)^{\beta}} - 5\cdot e^{-4\left(\frac{t}{\eta}\right)^{\beta}} + 2\cdot e^{-5\left(\frac{t}{\eta}\right)^{\beta}}$, or:

$$\lambda_s(t)=\frac{4\cdot\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}e^{-2\left(\frac{t}{\eta}\right)^{\beta}}+6\cdot\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}e^{-3\left(\frac{t}{\eta}\right)^{\beta}}}{2\cdot e^{-2\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-3\left(\frac{t}{\eta}\right)^{\beta}}-5\cdot e^{-4\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-5\left(\frac{t}{\eta}\right)^{\beta}}}+\frac{-20\cdot\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}e^{-4\left(\frac{t}{\eta}\right)^{\beta}}+10\cdot\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}e^{-5\left(\frac{t}{\eta}\right)^{\beta}}}{2\cdot e^{-2\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-3\left(\frac{t}{\eta}\right)^{\beta}}-5\cdot e^{-4\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-5\left(\frac{t}{\eta}\right)^{\beta}}}$$

The corresponding plot is given below.



The $MTTF$ of the system is obtained by integrating the system's reliability function given by $R_s(t)=2\cdot e^{-2\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-3\left(\frac{t}{\eta}\right)^{\beta}}-5\cdot e^{-4\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-5\left(\frac{t}{\eta}\right)^{\beta}}$ from time zero to infinity, as given by $MTTF=\int_0^{\infty}R_s\left(t\right)dt$. Using BlockSim's Analytical QCP, an $MTTF$ of 1007.8 hours is calculated, as shown in the figure below.

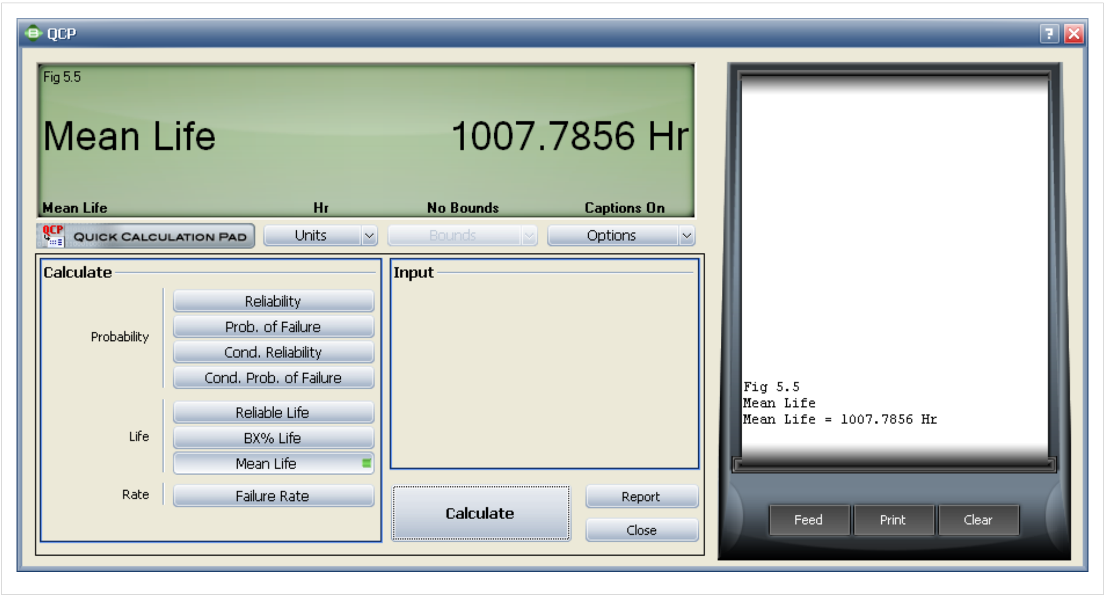


Fig 5.5

Mean Life 1007.7856 Hr

Mean Life Hr No Bounds Captions On

```
Fig 5.5
Mean Life
Mean Life = 1007.7856 Hr
```

The warranty time can be obtained by solving $R_s(t)$ with respect to time for a system reliability $R_s=0.9$. Using the Analytical QCP and selecting the **Reliable Life** option, a time of 372.72 hours is obtained, as shown in the following figure.

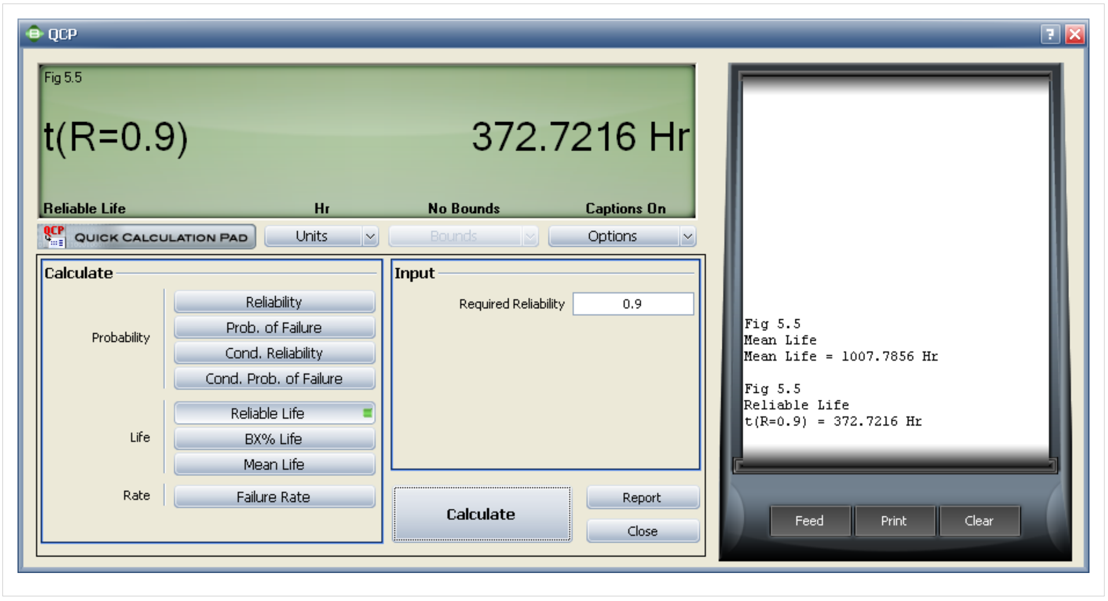


Fig 5.5

t(R=0.9) 372.7216 Hr

Reliable Life Hr No Bounds Captions On

Required Reliability 0.9

```
Fig 5.5
Mean Life
Mean Life = 1007.7856 Hr

Fig 5.5
Reliable Life
t(R=0.9) = 372.7216 Hr
```

Lastly, the conditional reliability can be obtained using $R(T,t)=\dfrac{R(T+t)}{R(T)}$ and $R_s(t)=2\cdot e^{-2\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-3\left(\frac{t}{\eta}\right)^{\beta}}-5\cdot e^{-4\left(\frac{t}{\eta}\right)^{\beta}}+2\cdot e^{-5\left(\frac{t}{\eta}\right)^{\beta}}$, or:

$$
\begin{aligned}
R(200,200) =& \frac{R(400)}{R(200)} \\
=& \frac{0.883825}{0.975321} \\
=& 0.906189
\end{aligned}
$$

This can be calculated using BlockSim's Analytical QCP, as shown below.

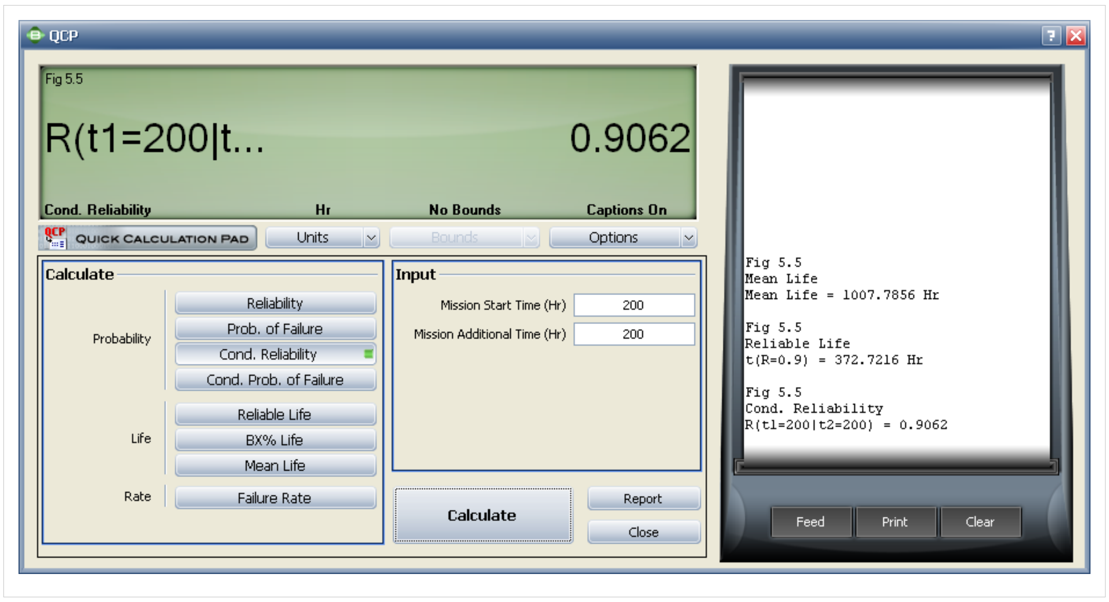


Fig 5.5

R(t1=200|t... 0.9062

Cond. Reliability Hr No Bounds Captions On

Mission Start Time (Hr) 200

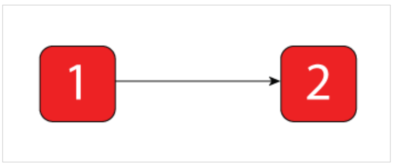
Mission Additional Time (Hr) 200

```
Fig 5.5
Mean Life
Mean Life = 1007.7856 Hr

Fig 5.5
Reliable Life
t(R=0.9) = 372.7216 Hr

Fig 5.5
Cond. Reliability
R(t1=200|t2=200) = 0.9062
```

Approximating the System cdf

In many cases, it is valuable to fit a distribution that represents the system's times-to-failure. This can be useful when the system is part of a larger assembly and may be used for repeated calculations or in calculations for other systems. In cases such as this, it can be useful to characterize the system's behavior by fitting a distribution to the overall system and calculating parameters for this distribution. This is equivalent to fitting a single distribution to describe $R_S(t)$. In essence, it is like reducing the entire system to a component in order to simplify calculations.

For the system shown below:



$$R_S(t)=e^{-\frac{1}{10{,}000}t}\cdot e^{-\left(\frac{t}{10{,}000}\right)^{6}}$$

To compute an approximate reliability function for this system, $R_A(t)\simeq R_S(t)$, one would compute $n$pairs of reliability and time values and then fit a single distribution to the data, or:

$$R_S(t=10{,}396.7)=10\%$$
$$R_S(t=9{,}361.9)=20\%$$
$$...$$
$$R_S(t=1{,}053.6)=90\%$$

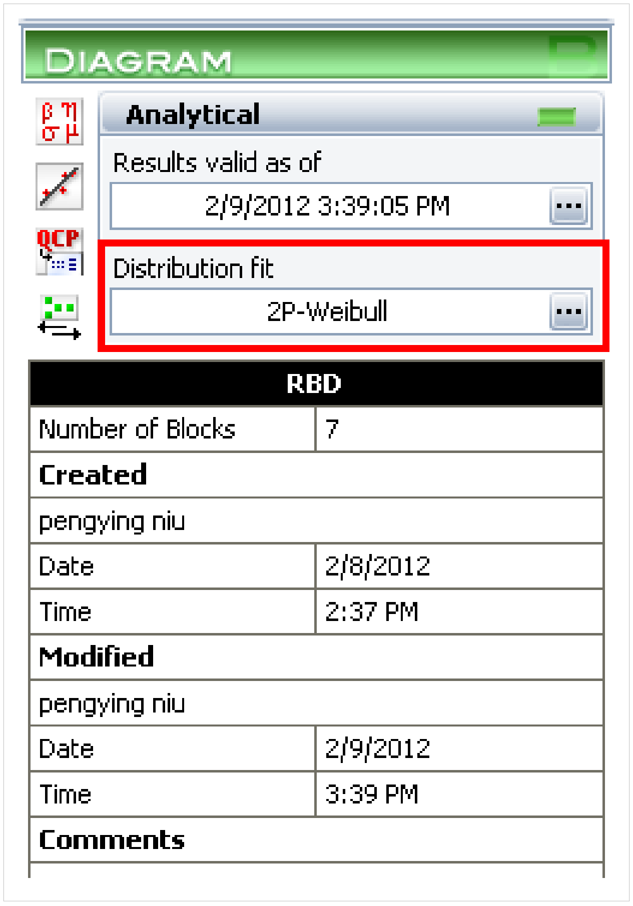
A single distribution, $R_A(t)$, that approximates $R_S(t)$can now be computed from these pairs using life data analysis methods. If using the Weibull++ software, one would enter the values as free-form data.

Example

Compute a single Weibull distribution approximation for the parallel system in the previous example.

Solution

Assume that the system can be approximated by a 2-parameter Weibull distribution with $\beta=2.02109$and $\eta=1123.51$. In BlockSim, this is accomplished by representing the entire system as one distribution. To do this, click the **Distribution fit** area on the diagram's control panel, as shown next.



DIAGRAM

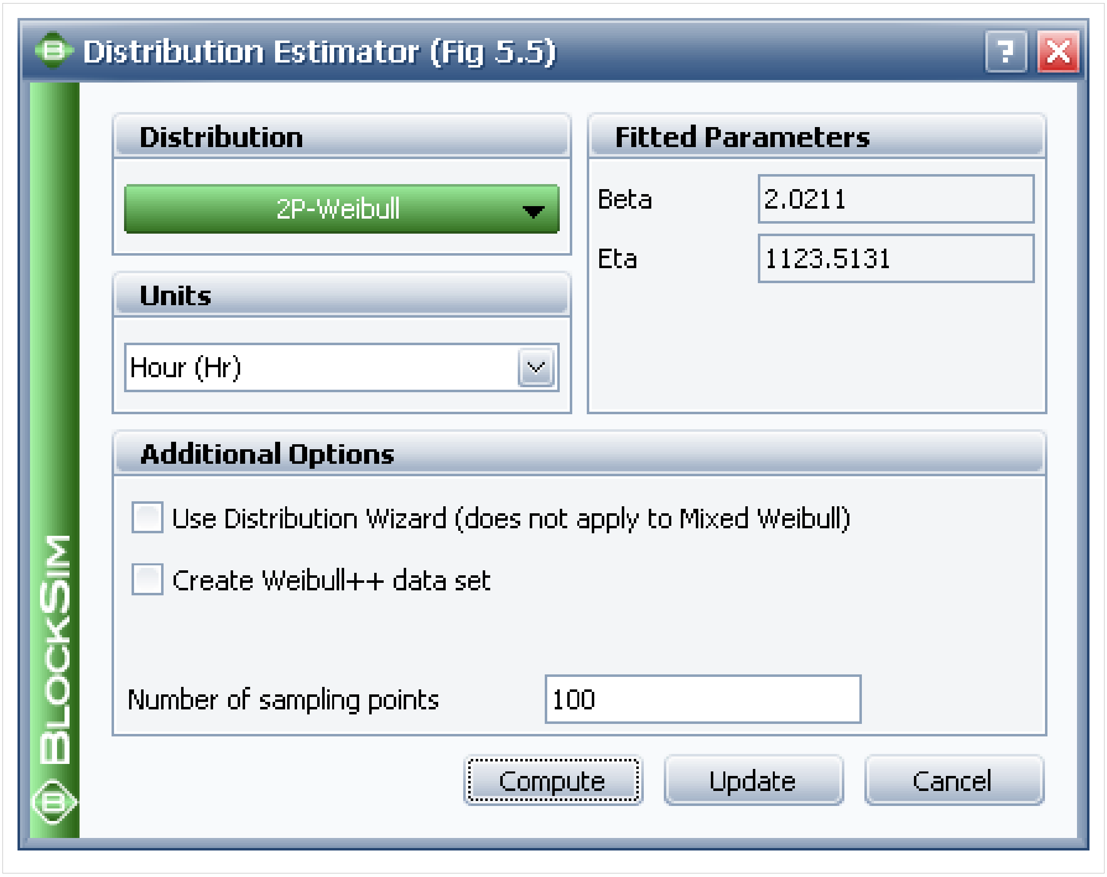
Analytical

Results valid as of: 2/9/2012 3:39:05 PM

Distribution fit: 2P-Weibull

RBD	
Number of Blocks	7
Created	
pengying niu	
Date	2/8/2012
Time	2:37 PM
Modified	
pengying niu	
Date	2/9/2012
Time	3:39 PM
Comments	

This opens the Distribution Estimator window, which allows you to select a distribution to represent the data.



Distribution Estimator (Fig 5.5)

Distribution: 2P-Weibull

Units: Hour (Hr)

Fitted Parameters

Beta	2.0211
Eta	1123.5131

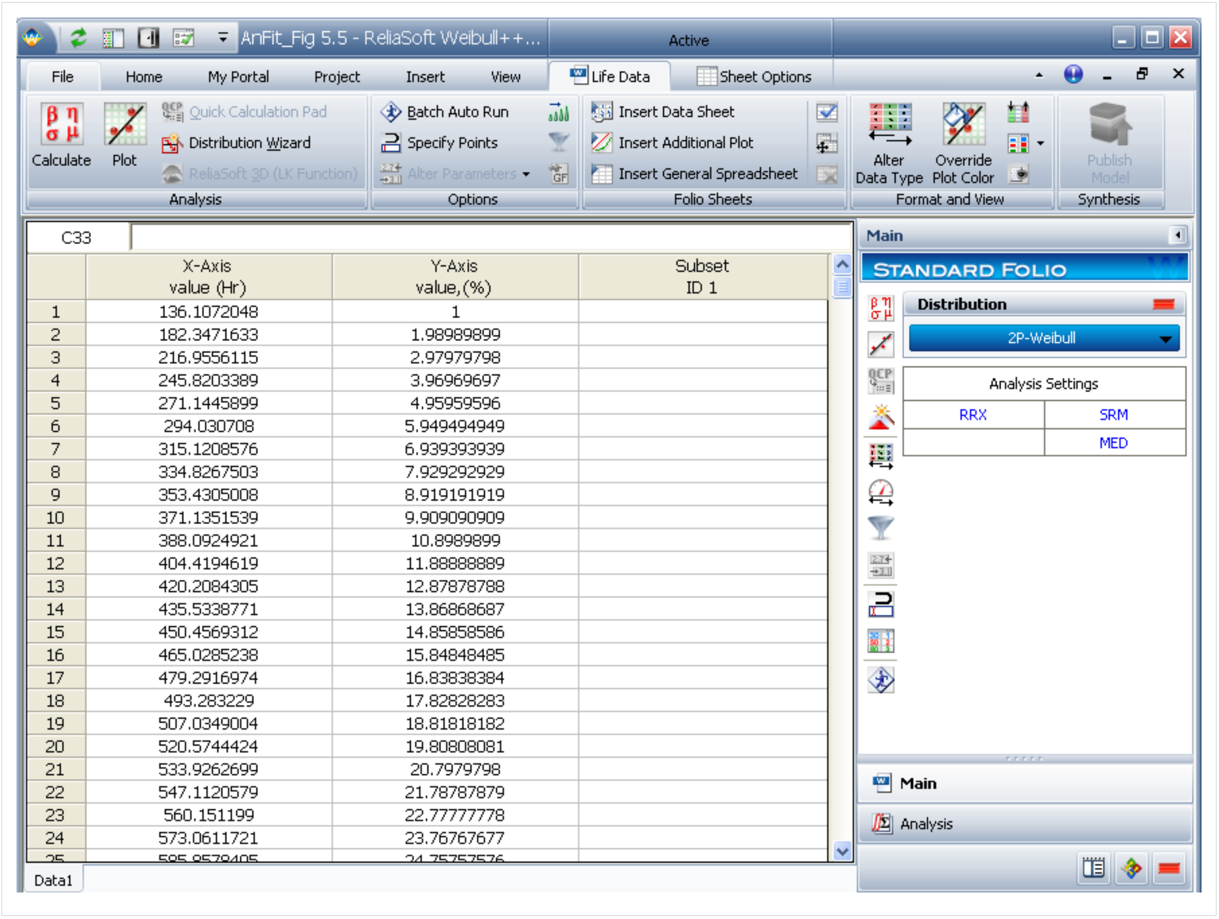
Additional Options

- [ ] Use Distribution Wizard (does not apply to Mixed Weibull)
- [ ] Create Weibull++ data set

Number of sampling points: 100

When you analyze the diagram, BlockSim will generate a number of system failure times based on the system's reliability function. The system's reliability function can be used to solve for a time value associated with an unreliability value. For example, consider an unreliability value of $F(t) = 0.11$. Using the system's reliability equation, the corresponding time-to-failure for a 0.11 unreliability is 389.786 hours. The distribution of the generated time values can then be fitted to a probability distribution function.

When enough points have been generated, the selected distribution will be fitted to the generated data set and the distribution's parameters will be calculated. In addition, if ReliaSoft's Weibull++ is installed, the fit of the distribution can be analyzed using a Weibull++ standard folio, as shown in the next figure. It is recommended that the analyst examine the fit to ascertain the applicability of the approximation.



	X-Axis value (Hr)	Y-Axis value,(%)	Subset ID 1
1	136.1072048	1	
2	182.3471633	1.98989899	
3	216.9556115	2.97979798	
4	245.8203389	3.96969697	
5	271.1445899	4.95959596	
6	294.030708	5.949494949	
7	315.1208576	6.939393939	
8	334.8267503	7.929292929	
9	353.4305008	8.919191919	
10	371.1351539	9.909090909	
11	388.0924921	10.8989899	
12	404.4194619	11.88888889	
13	420.2084305	12.87878788	
14	435.5338771	13.86868687	
15	450.4569312	14.85858586	
16	465.0285238	15.84848485	
17	479.2916974	16.83838384	
18	493.283229	17.82828283	
19	507.0349004	18.81818182	
20	520.5744424	19.80808081	
21	533.9262699	20.7979798	
22	547.1120579	21.78787879	
23	560.151199	22.77777778	
24	573.0611721	23.76767677	
25	585.8578405	24.75757576	

Main

STANDARD FOLIO

Distribution

2P-Weibull

Analysis Settings

RRX	SRM
	MED

Duty Cycle

Components of a system may not operate continuously during a system's mission, or may be subjected to loads greater or lesser than the rated loads during system operation. To model this, a factor called the Duty Cycle (d_c) is used. The duty cycle may also be used to account for changes in environmental stress, such as temperature changes, that may effect the operation of a component. The duty cycle is a positive value, with a default value of 1 representing continuous operation at rated load, and any values other than 1 representing other load values with respect to the rated load value (or total operating time). A duty cycle value higher than 1 indicates a load in excess of the rated value. A duty cycle value lower than 1 indicates that the component is operating at a load lower than the rated load or not operating continuously during the system's mission. For instance, a duty cycle of 0.5 may be used for a component that operates only half of the time during the system's mission.

The reliability metrics for a component with a duty cycle are calculated as follows. Let $d_c$represent the duty cycle during a particular mission of the component, $t$represent the mission time and t'represent the accumulated age. Then:

$$t' = d_c \times t$$

The reliability equation for the component is:

$$R(t') = R(d_c \times t)$$

The component *pdf* is:

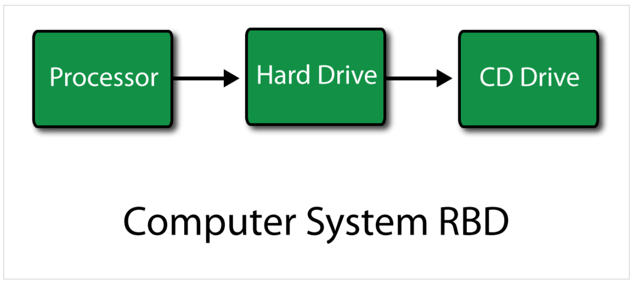
$$f(t') = -\frac{d(R(t'))}{dt} = -\frac{d(R(d_c \times t))}{dt} = d_c f(d_c \times t)$$

The failure rate of the component is:

$$\lambda(t') = \frac{f(t')}{R(t')} = \frac{d_c f(d_c \times t)}{R(d_c \times t)} = d_c \lambda(d_c \times t)$$

Example

Consider a computer system with three components: a processor, a hard drive and a CD drive in series as shown next. Assume that all three components follow a Weibull failure distribution. The processor has the following parameters, $\beta_1 = 1.5$and $\eta_1 = 5000$. For the hard drive, the parameters are $\beta_2 = 2.5$and $\eta_2 = 3000$, and for the CD drive they are $\beta_3 = 2$and $\eta_3 = 4000$. Determine the reliability of the computer system after one year (365 days) of operation, assuming that the CD drive is used only 30% of the time.



Solution

The reliability of the processor after 365 days of operation is given by:

$$\begin{aligned} R_{processor}(365) &= e^{-\left(\frac{365}{\eta_1}\right)^{\beta_1}} \\ &= e^{-\left(\frac{365}{5000}\right)^{1.5}} \\ &= 0.9805 \text{ or } 98.05\% \end{aligned}$$

The reliability of the hard drive after 365 days of operation is given by:

$$\begin{aligned} R_{harddrive}(365) &= e^{-\left(\frac{365}{\eta_2}\right)^{\beta_2}} \\ &= e^{-\left(\frac{365}{3000}\right)^{2.5}} \\ &= 0.9948 \text{ or } 99.48\% \end{aligned}$$

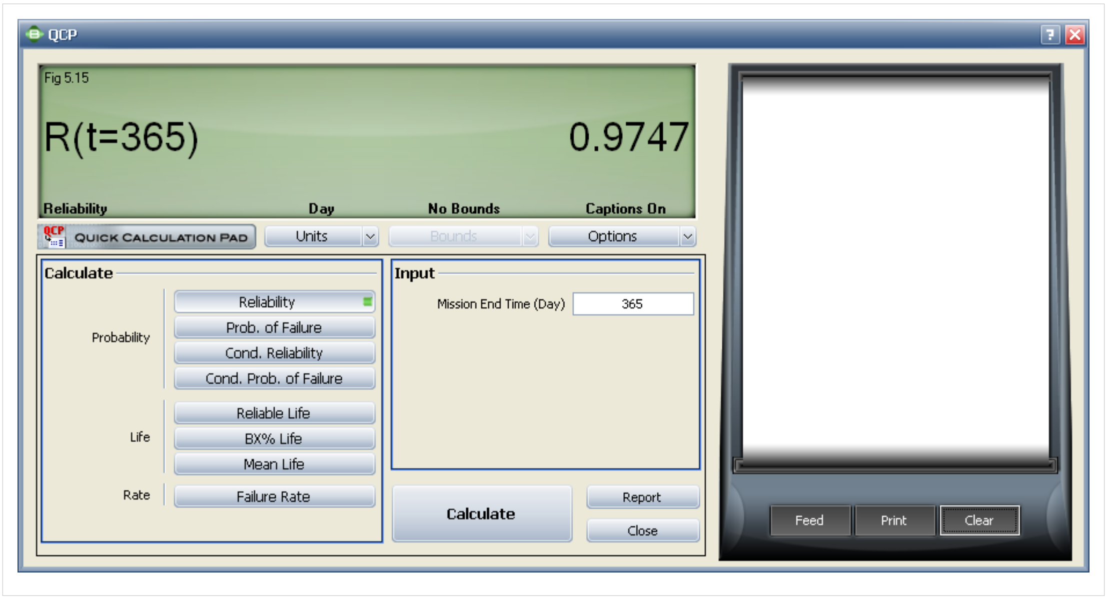
The reliability of the CD drive after 365 days of operation (taking into account the 30% operation using a duty cycle of 0.3) is given by:

$$\begin{aligned} R_{CDdrive}(365) &= e^{-\left(\frac{d_c \times 365}{\eta_3}\right)^{\beta_3}} \\ &= e^{-\left(\frac{0.3 \times 365}{4000}\right)^{2}} \\ &= 0.9993 \text{ or } 99.93\% \end{aligned}$$

Thus the reliability of the computer system after 365 days of operation is:

$$\begin{aligned} R_s(365) =& R_{processor}(365)\cdot R_{harddrive}(365)\cdot R_{CDdrive}(365) \\ =& 0.9805\cdot 0.9948\cdot 0.9993 \\ =& 0.9747 \text{ or } 97.47\% \end{aligned}$$

The result can also be obtained in BlockSim by creating an RBD of the system and using the Quick Calculation Pad (QCP) to calculate the reliability, as shown in the following figure.

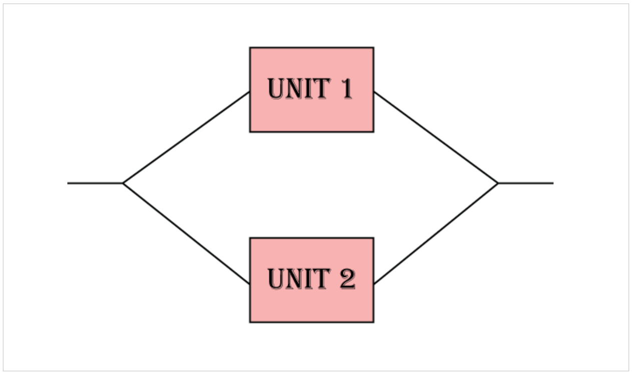


Load Sharing

As presented in earlier chapters, a reliability block diagram (RBD) allows you to graphically represent how the components within a system are reliability-wise connected. In most cases, independence is assumed across the components within the system. For example, the failure of component A does not affect the failure of component B. However, if a system consists of components that are sharing a load, then the assumption of independence no longer holds true.

If one component fails, then the component(s) that are still operating will have to assume the failed unit's portion of the load. Therefore, the reliabilities of the surviving unit(s) will change. Calculating the system reliability is no longer an easy proposition. In the case of load sharing components, the change of the failure distributions of the surviving components must be known in order to determine the system's reliability.

To illustrate this, consider a system of two units connected reliability-wise in parallel as shown below.



Assume that the units must supply an output of 8 volts and that if both units are operational, each unit is to supply 50% of the total output. If one of the units fails, then the surviving unit supplies 100%. Furthermore, assume that having to supply the entire load has a negative impact on the reliability characteristics of the surviving unit.

Because the reliability characteristics of the unit change based on the load it is sharing, a method that can model the effect of the load on life should be used. One way to do this is to use a life distribution along with a life-stress relationship (as discussed in A Brief Introduction to Life-Stress Relationships) for each component. The detailed discussion for this method can be found at Additional Information on Load Sharing. Another simple way is to use the concept of acceleration factors and assume that the load has a linear effect on the failure time. If the load is doubled, then the life of the component will be shortened by half.

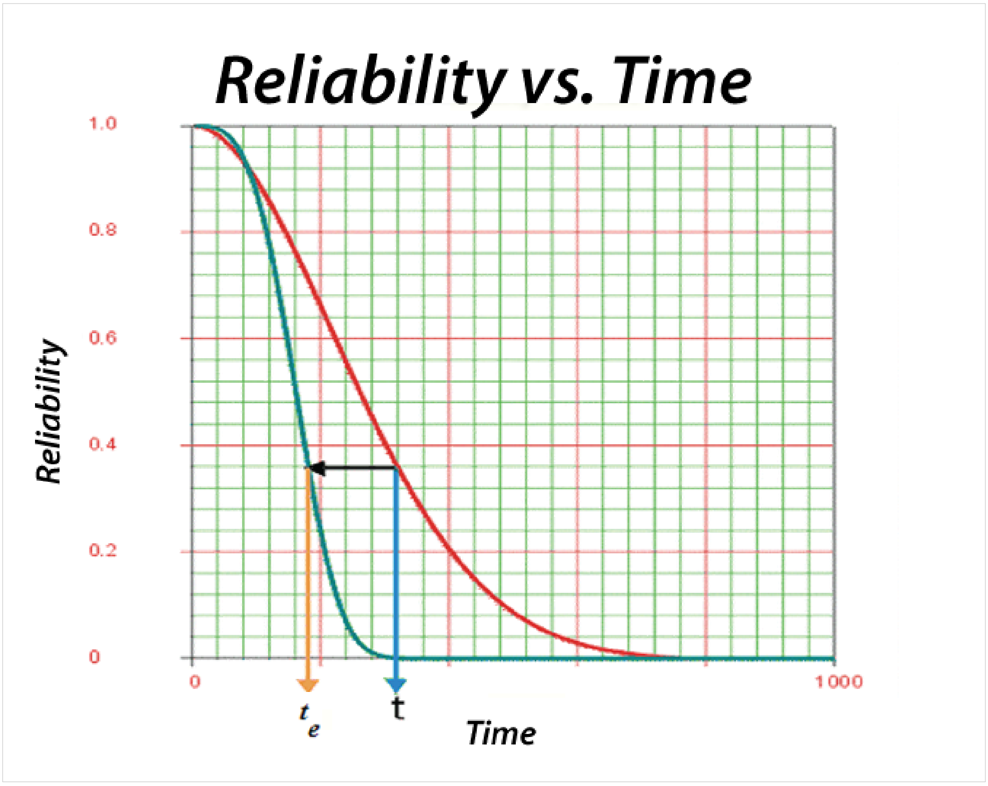
For the above load sharing system, the reliability of each component is a function of time and load. For example, for Unit 1, the reliability and the probability density function are:

$$R_1(t, S_1) \text{and } f_1(t, S_1)$$

where $S_1$is the load shared by Unit 1 at time *t* and the total load of the system is $S = S_1 + S_2$. At the beginning, both units are working. Assume that Unit 1 fails at time *x* and Unit 2 takes over the entire load. The reliability for Unit 2 at time *x* is:

$$R_2(x, S_2) = R_2(t_{2e}, S)$$
$$t_{2e} = \frac{S_2}{S}x$$

$t_{2e}$is the equivalent time for Unit 2 at time *x* if it is operated with load *S*. The equivalent time concept is illustrated in the following plot.



The system reliability at time *t* is:

$$R(t) = R_1(t, S_1) \cdot R_2(t, S_2)$$
$$+ \int_0^t f_1(x, S_1) \cdot R_2(t_{2e} + (t - x), S)dx$$
$$+ \int_0^t f_2(x, S_2) \cdot R_1(t_{1e} + (t - x), S)dx$$

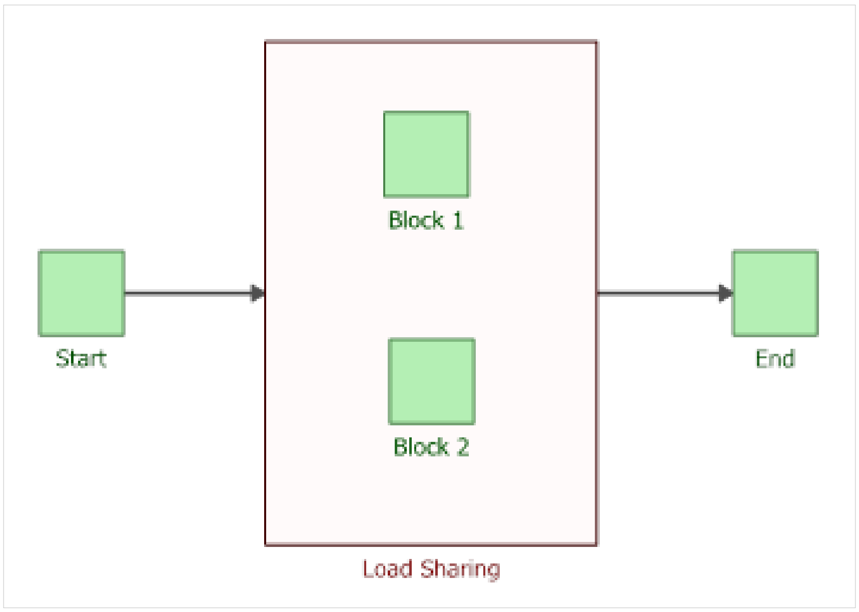
In BlockSim, the failure time distribution for each component is defined at the load of *S*. The reliability function for a component at a given load is calculated as:

$$R_i(t, S_i) = R_i(t \times \frac{S_i}{S}, S)$$
$$f_i(t, S_i) = \frac{S_i}{S} f_i\left(t \times \frac{S_i}{S}, S\right)$$

From the above equation, it can be seen that the concept used in the calculation for load sharing is the same as the concept used in the calculation for duty cycle.

Example

In the following load sharing system, Block 1 follows a Weibull failure distribution with $\beta_1 = 1.5$, and $\eta_1 = 1,000$. Block 2 follows a Weibull failure distribution with $\beta_2 = 2$, and $\eta_2 = 2,000$. The load for Block 1 is 1 unit, and for Block 2 it is 3 units. Calculate the system reliability at time 1,500.



Block 1 shares 25% (P1) of the entire load, and Block 2 shares 75% (P2) of it. Therefore, we have the following equations for calculating the system reliability:

$$R_i(t, S_i) = R_i(P_i \times t),$$
$$f_i(x, S_i) = P_i f_i(P_i \times x)$$

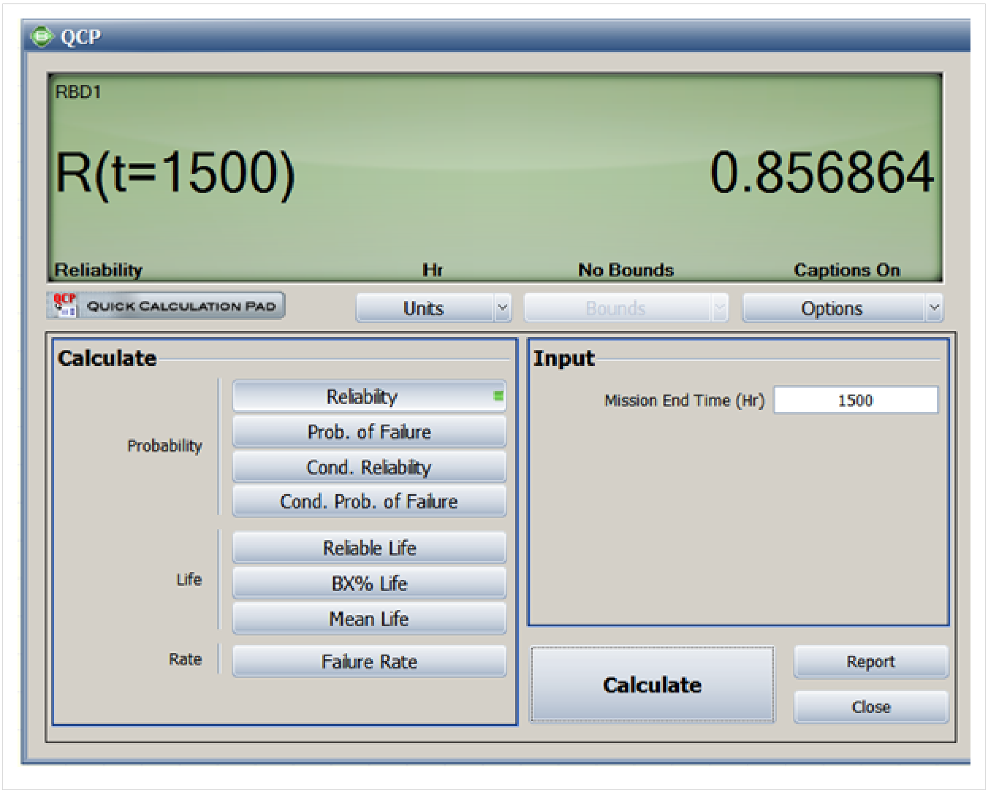
and:

$$R_1(t_{1e} + (t - x)) = R_1(P_1 x + t - x) = R_1(t - P_2 x)$$
$$R_2(t_{2e} + (t - x)) = R_2(P_2 x + t - x) = R_2(t - P_1 x)$$

Using the above equations in the system reliability function, we get:

$$R(t) = R_1(t, S_1) \cdot R_2(t, S_2) + \int_0^t f_1(x, S_1) \cdot R_2(t_{2e} + (t - x), S) dx$$
$$+ \int_0^t f_2(x, S_2) \cdot R_1(t_{1e} + (t - x), S) dx$$
$$= R_1(P_1 \times t) \cdot R_2(P_2 \times t) + \int_0^t P_1 f_1(P_1 x) \cdot R_2(t - P_1 x) dx$$
$$+ \int_0^t P_2 f_2(P_2 x) \cdot R_1(t - P_2 x) dx$$

The calculated system reliability at time 1,500 is 0.8569, as given below.


QCP

RBD1

R(t=1500) 0.856864

Reliability Hr No Bounds Captions On

Mission End Time (Hr): 1500

Standby Components

In the previous section, the case of a system with load sharing components was presented. This is a form of redundancy with dependent components. That is, the failure of one component affects the failure of the other(s). This section presents another form of redundancy: standby redundancy. In standby redundancy the redundant components are set to be under a lighter load condition (or no load) while not needed and under the operating load when they are activated.

In standby redundancy the components are set to have two states: an active state and a standby state. Components in standby redundancy have two failure distributions, one for each state. When in the standby state, they have a quiescent (or dormant) failure distribution and when operating, they have an active failure distribution.

In the case that both quiescent and active failure distributions are the same, the units are in a simple parallel configuration (also called a hot standby configuration). When the rate of failure of the standby component is lower in quiescent mode than in active mode, that is called a warm standby configuration. When the rate of failure of the standby component is zero in quiescent mode (i.e., the component cannot fail when in standby), that is called a cold standby configuration.

Simple Standby Configuration

Consider two components in a standby configuration. Component 1 is the active component with a Weibull failure distribution with parameters $\beta = 1.5$and $\eta = 1,000$. Component 2 is the standby component. When Component 2 is operating, it also has a Weibull failure distribution with $\beta = 1.5$and $\eta = 1,000$. Furthermore, assume the following cases for the quiescent distribution.

- Case 1: The quiescent distribution is the same as the active distribution (hot standby).
- Case 2: The quiescent distribution is a Weibull distribution with $\beta = 1.5$and $\eta = 2000$(warm standby).
- Case 3: The component cannot fail in quiescent mode (cold standby).

In this case, the reliability of the system at some time, t, can be obtained using the following equation:

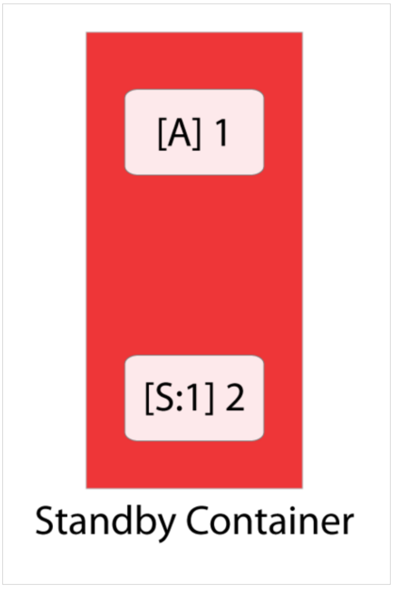
$$R(t) = R_1(t) + \int_0^t f_1(x) \cdot R_{2;SB}(x) \cdot \frac{R_{2;A}(t_e + t - x)}{R_{2;A}(t_e)} dx$$

where:

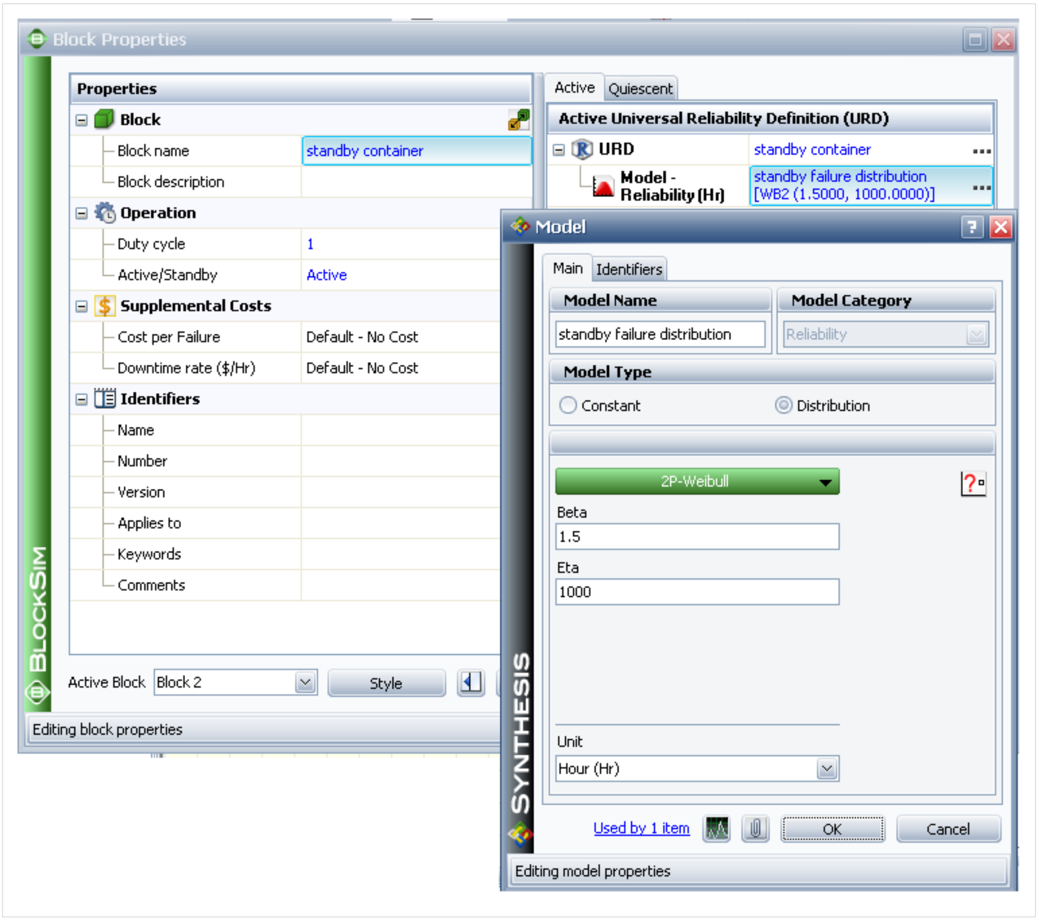
- $R_1$is the reliability of the active component.
- $f_1$is the *pdf* of the active component.
- $R_{2;SB}$is the reliability of the standby component when in standby mode (quiescent reliability).
- $R_{2;A}$is the reliability of the standby component when in active mode.
- $t_e$is the equivalent operating time for the standby unit if it had been operating at an active mode, such that:

$$R_{2;SB}(x) = R_{2;A}(t_e)$$

The second equation above can be solved for $t_e$and substituted into the first equation above. The following figure illustrates the example as entered in BlockSim using a standby container.



The active and standby blocks are within a container, which is used to specify standby redundancy. Since the standby component has two distributions (active and quiescent), the Block Properties window of the standby block has two pages for specifying each one. The following figures illustrate these pages.


Block Properties

Properties

Block

Block name	standby container
Block description	

Operation

Duty cycle	1
Active/Standby	Active

Supplemental Costs

Cost per Failure	Default - No Cost
Downtime rate ($/Hr)	Default - No Cost

Identifiers

Name	
Number	
Version	
Applies to	
Keywords	
Comments	

Active Block: Block 2

Editing block properties

Active | Quiescent

Active Universal Reliability Definition (URD)

URD	standby container
Model - Reliability (Hr)	standby failure distribution [WB2 (1.5000, 1000.0000)]

Model

Main | Identifiers

Model Name: standby failure distribution

Model Category: Reliability

Model Type: Constant [✓ Distribution]

2P-Weibull

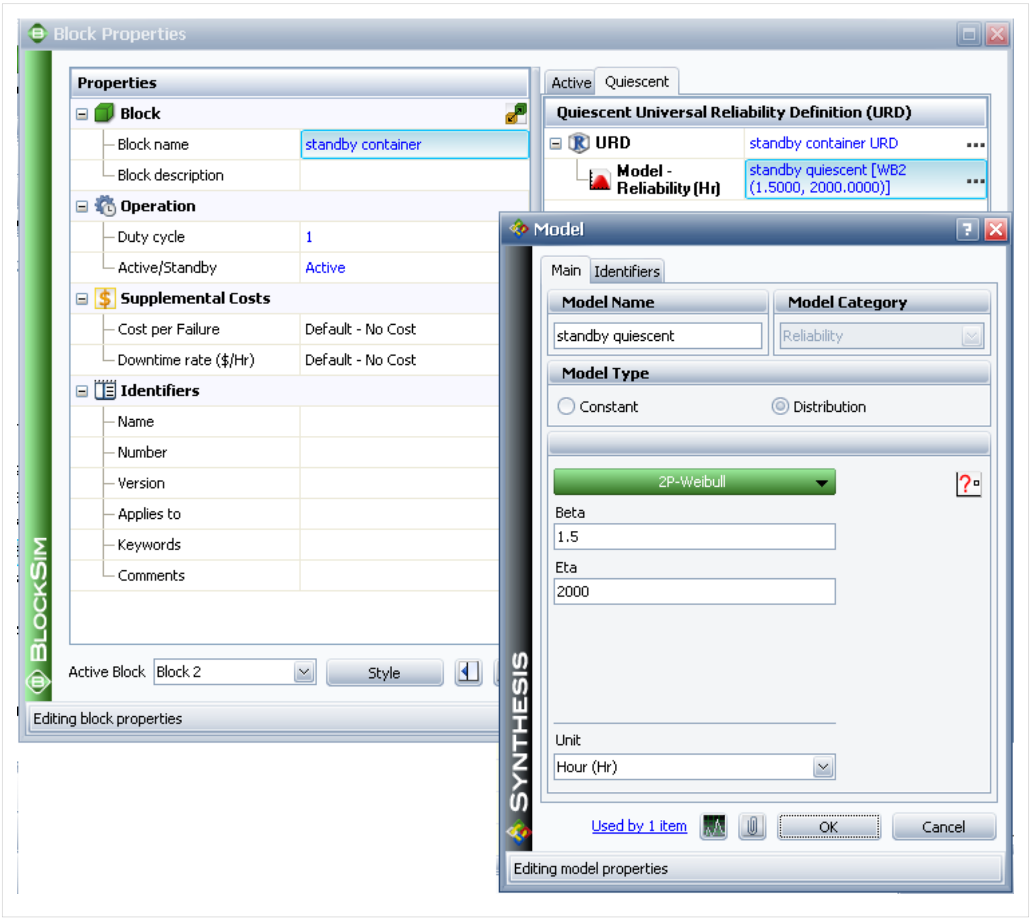
Beta: 1.5

Eta: 1000

Unit: Hour (Hr)

Used by 1 item

Editing model properties



Block	
Block name	standby container
Block description	
Operation	
Duty cycle	1
Active/Standby	Active
Supplemental Costs	
Cost per Failure	Default - No Cost
Downtime rate ($/Hr)	Default - No Cost
Identifiers	
Name	
Number	
Version	
Applies to	
Keywords	
Comments	

Active Block: Block 2

Editing block properties

Quiescent Universal Reliability Definition (URD)	
URD	standby container URD
Model - Reliability (Hr)	standby quiescent [WB2 (1.5000, 2000.0000)]

Model Name: standby quiescent

Model Category: Reliability

Model Type: Constant [✓ Distribution]

2P-Weibull

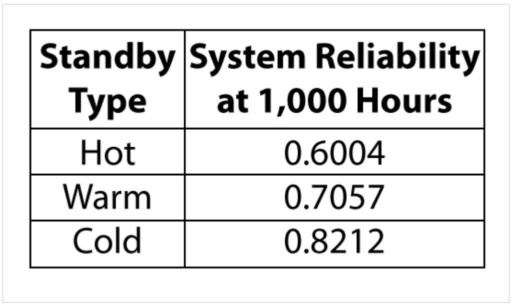
Beta: 1.5

Eta: 2000

Unit: Hour (Hr)

Editing model properties

The system reliability results for 1000 hours are given in the following table:



Standby Type	System Reliability at 1,000 Hours
Hot	0.6004
Warm	0.7057
Cold	0.8212

Note that even though the β value for the quiescent distribution is the same as in the active distribution, it is possible that the two can be different. That is, the failure modes present during the quiescent mode could be different from the modes present during the active mode. In that sense, the two distribution types can be different as well (e.g., lognormal when quiescent and Weibull when active).

In many cases when considering standby systems, a switching device may also be present that switches from the failed active component to the standby component. The reliability of the switch can also be incorporated into

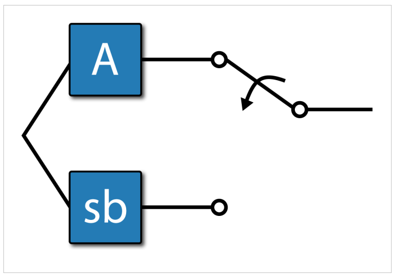
$$R(t)=R_1(t)+\int_0^t f_1(x)\cdot R_{2;SB}(x)\cdot\frac{R_{2;A}(t_e+t-x)}{R_{2;A}(t_e)}dx$$

as presented in the next section.

BlockSim's System Reliability Equation window returns a single token for the reliability of units in a standby configuration. This is the same as the load sharing case presented in the previous section.

Reliability of Standby Systems with a Switching Device

In many cases when dealing with standby systems, a switching device is present that will switch to the standby component when the active component fails. Therefore, the failure properties of the switch must also be included in the analysis.



In most cases when the reliability of a switch is to be included in the analysis, two probabilities can be considered. The first and most common one is the probability of the switch performing the action (i.e., switching) when requested to do so. This is called Switch Probability per Request in BlockSim and is expressed as a static probability (e.g., 90%). The second probability is the quiescent reliability of the switch. This is the reliability of the switch as it ages (e.g., the switch might wear out with age due to corrosion, material degradation, etc.). Thus it is possible for the switch to fail before the active component fails. However, a switch failure does not cause the system to fail, but rather causes the system to fail only if the switch is needed and the switch has failed. For example, if the active component does not fail until the mission end time and the switch fails, then the system does not fail. However, if the active component fails and the switch has also failed, then the system cannot be switched to the standby component and it therefore fails.

In analyzing standby components with a switching device, either or both failure probabilities (during the switching or while waiting to switch) can be considered for the switch, since each probability can represent different failure modes. For example, the switch probability per request may represent software-related issues or the probability of detecting the failure of an active component, and the quiescent probability may represent wear-out type failures of the switch.

To illustrate the formulation, consider the previous example that assumes perfect switching. To examine the effects of including an imperfect switch, assume that when the active component fails there is a 90% probability that the switch will switch from the active component to the standby component. In addition, assume that the switch can also fail due to a wear-out failure mode described by a Weibull distribution with $\beta = 1.7$and $\eta = 5000$.

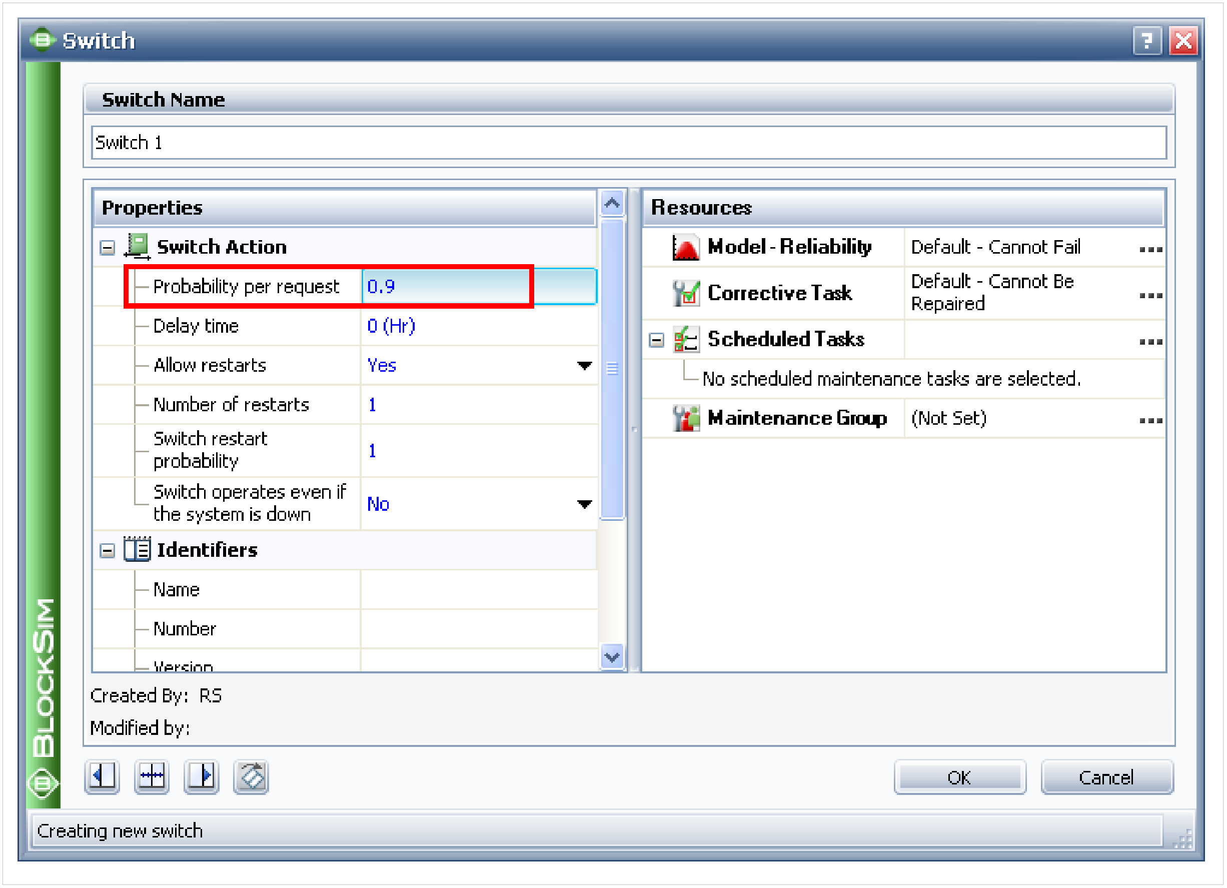
Therefore, the reliability of the system at some time, t, is given by the following equation.

$$\begin{aligned} R(t) = & R_1(t) + \int_0^t \{f_1(x) \cdot R_{2;SB}(x) \\ & \cdot \frac{R_{2;A}(t_e + t - x)}{R_{2;A}(t_e)} \cdot R_{SW;Q}(x) \cdot R_{SW;REQ}(x)\} dx \end{aligned}$$

where:

- $R_1$is the reliability of the active component.
- $f_1$is the *pdf* of the active component.
- $R_{2;SB}$is the reliability of the standby component when in standby mode (quiescent reliability).
- $R_{2;A}$is the reliability of the standby component when in active mode.
- $R_{SW;Q}$is the quiescent reliability of the switch.
- $R_{SW;REQ}$is the switch probability per request.
- $t_e$is the equivalent operating time for the standby unit if it had been operating at an active mode.

This problem can be solved in BlockSim by including these probabilities in the container's properties, as shown in the figures below. In BlockSim, the standby container is acting as the switch.



Switch Name: Switch 1

Properties

Switch Action

Probability per request	0.9
Delay time	0 (Hr)
Allow restarts	Yes
Number of restarts	1
Switch restart probability	1
Switch operates even if the system is down	No

Identifiers

Name	
Number	
Version	

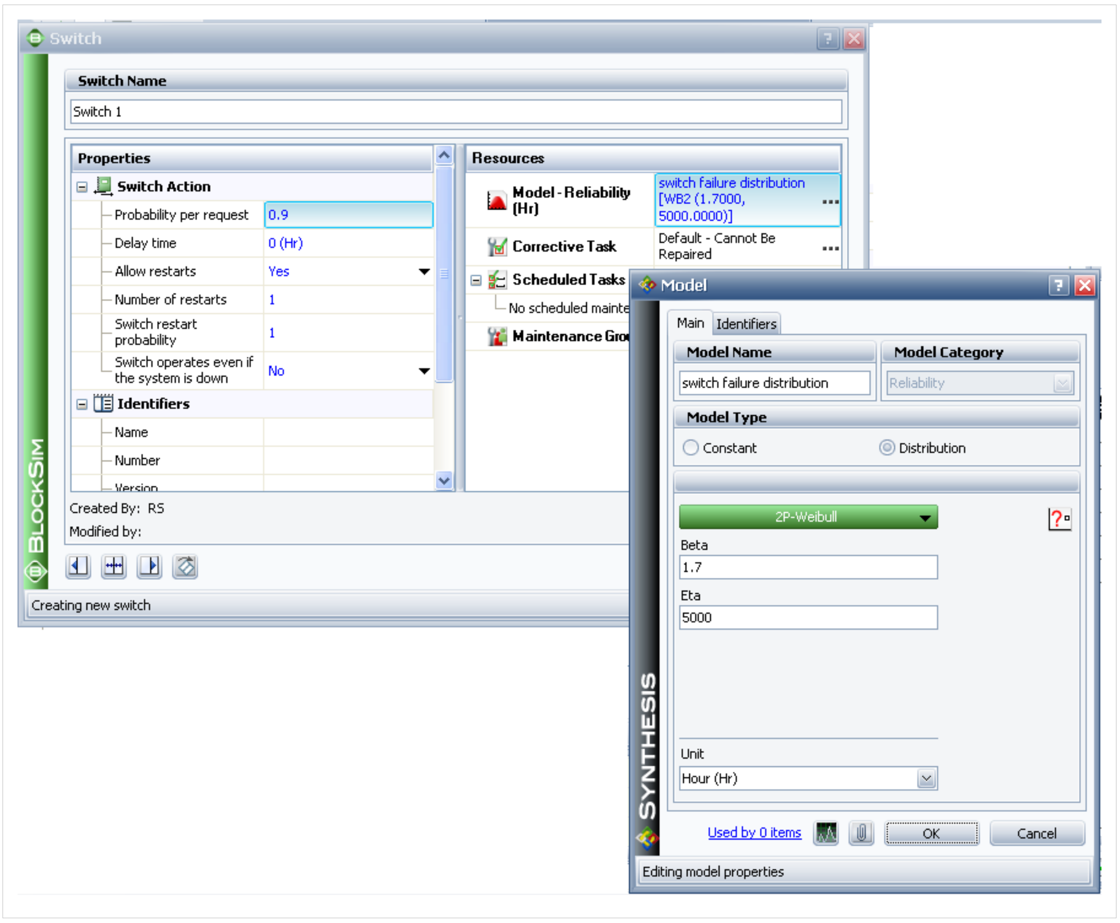
Resources

Model - Reliability	Default - Cannot Fail
Corrective Task	Default - Cannot Be Repaired
Scheduled Tasks	
No scheduled maintenance tasks are selected.	
Maintenance Group	(Not Set)

Created By: RS

Modified by:

Creating new switch


Switch

Switch Name: Switch 1

Properties	
Switch Action	
Probability per request	0.9
Delay time	0 (Hr)
Allow restarts	Yes
Number of restarts	1
Switch restart probability	1
Switch operates even if the system is down	No
Identifiers	
Name	
Number	
Version	

Resources	
Model - Reliability (Hr)	switch failure distribution [WB2 (1.7000, 5000.0000)]
Corrective Task	Default - Cannot Be Repaired
Scheduled Tasks	No scheduled mainte
Maintenance Gro	

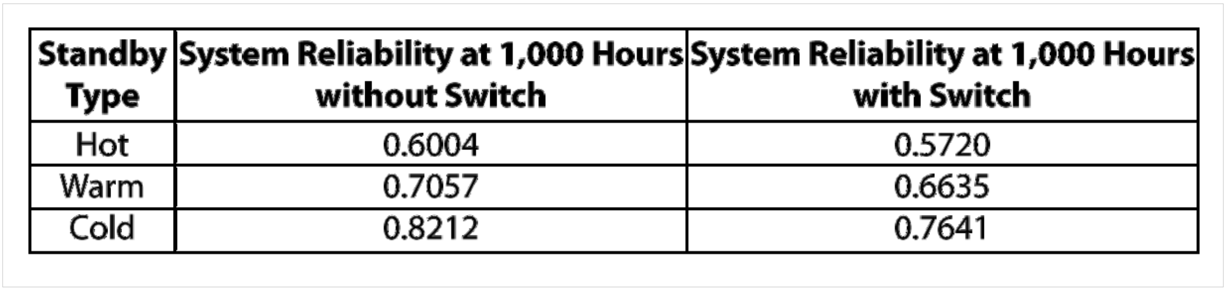
Created By: RS
Modified by:
Creating new switch

Model
Main | Identifiers
Model Name: switch failure distribution
Model Category: Reliability
Model Type: Constant [✓ Distribution]
2P-Weibull
Beta: 1.7
Eta: 5000
Unit: Hour (Hr)
Used by 0 items
OK | Cancel
Editing model properties

Note that there are additional properties that can be specified in BlockSim for a switch, such as Switch Restart Probability, No. of Restarts and Switch Delay Time. In many applications, the switch is re-tested (or re-cycled) if it fails to switch the first time. In these cases, it might be possible that it switches in the second or third, or n^{th}attempt.

The Switch Restart Probability specifies each additional attempt's probability of successfully switching and the Finite Restarts specifies the total number of attempts. Note that the Switch Restart Probability specifies the probability of success of each trial (or attempt). The probability of success of $n$consecutive trials is calculated by BlockSim using the binomial distribution and this probability is then incorporated into equation above. The Switch Delay Time property is related to repairable systems and is considered in BlockSim only when using simulation. When using the analytical solution (i.e., for a non-repairable system), this property is ignored.

Solving the analytical solution (as given by the above equation), the following results are obtained.



Standby Type	System Reliability at 1,000 Hours without Switch	System Reliability at 1,000 Hours with Switch
Hot	0.6004	0.5720
Warm	0.7057	0.6635
Cold	0.8212	0.7641

From the table above, it can be seen that the presence of a switching device has a significant effect on the reliability of a standby system. It is therefore important when modeling standby redundancy to incorporate the switching device reliability properties. It should be noted that this methodology is not the same as treating the switching device as another series component with the standby subsystem. This would be valid only if the failure of the switch resulted in the failure of system (e.g., switch failing open). In equation above, the Switch Probability per Request and quiescent probability are present only in the second term of the equation. Treating these two failure modes as a series configuration with the standby subsystem would imply that they are also present when the active component is functioning (i.e., first term of equation above). This is invalid and would result in the underestimation of the reliability of the system. In other words, these two failure modes become significant only when the active component fails.

As an example, and if we consider the warm standby case, the reliability of the system without the switch is 70.57% at 1000 hours. If the system was modeled so that the switching device was in series with the warm standby subsystem, the result would have been:

$$\begin{aligned} R_S(1000) =& R_{Standby}(1000) \cdot R_{sw,Q(1000)} \cdot R_{sw,req} \\ =& 0.7057 \cdot 0.9372 \cdot 0.9 \\ =& 0.5952 \end{aligned}$$

In the case where a switch failure mode causes the standby subsystem to fail, then this mode can be modeled as an individual block in series with the standby subsystem.

Example

Consider a car with four new tires and a full-size spare. Assume the following failure characteristics:

- The tires follow a Weibull distribution with a $\beta = 4$and an $\eta = 40,000$miles while on the car due to wear.
- The tires also have a probability of failing due to puncture or other causes. For this, assume a constant rate for this occurrence with a probability of 1 every 50,000 miles.
- When not on the car (i.e., is a spare), a tire's probability of failing also has a Weibull distribution with a $\beta = 2$ and $\eta = 120,000$miles.

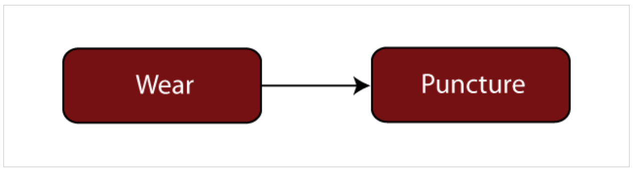
Assume a mission of 1,000 miles. If a tire fails during this trip, it will be replaced with the spare. However, the spare will not be repaired during the trip. In other words, the trip will continue with the spare on the car and, if the spare fails, the system will fail. Determine the probability of system failure.

Solution

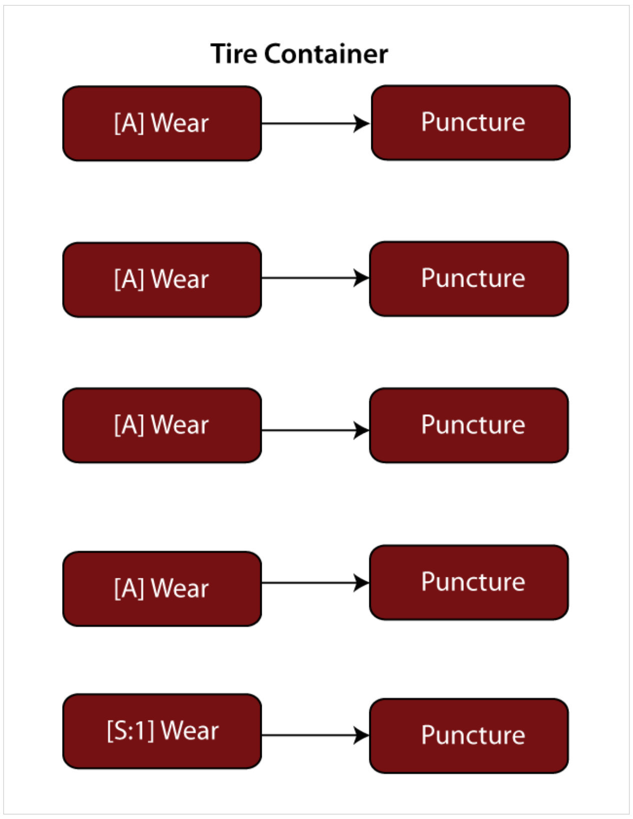
The active failure distribution for the tires are:

- Due to wearout, Weibull $\beta = 4$and $\eta = 40,000$miles.
- Due to random puncture, exponential $\mu = 50,000$.
- The quiescent failure distribution is a Weibull distribution with $\beta = 2$and $\eta = 120,000$miles.

The block diagram for each tire has two blocks in series, one block representing the wearout mode and the other the random puncture mode, as shown next:



There are five tires, four active and one standby (represented in the diagram by a standby container with a 4-out-of-5 requirement), as shown next:



For the standby Wear block, set the active failure and the quiescent distributions, but for the Puncture block, set only the active puncture distribution (because the tire cannot fail due to puncture while stored). Using BlockSim, the probability of system failure is found to be 0.003 or 0.3%.

See it in action...

More examples on load sharing and standby configurations are available! See also:

Modeling Failure Modes [1]

Load Sharing Configuration Example

Note Regarding Numerical Integration Solutions

Load sharing and standby solutions in BlockSim are performed using numerical integration routines. As with any numerical analysis routine, the solution error depends on the number of iterations performed, the step size chosen and related factors, plus the behavior of the underlying function. By default, BlockSim uses a certain set of preset factors. In general, these defaults are sufficient for most problems. If a higher precision or verification of the precision for a specific problem is required, BlockSim's preset options can be modified and/or the integration error can be assessed using the **Integration Parameters** option for each container. For more details, you can refer to the documentation on the Algorithm Setup window in the BlockSim help file.

References

[1] http://www.reliasoft.com/BlockSim/examples/rc4/index.htm

Chapter 5

Reliability Importance and Optimized Reliability Allocation (Analytical)

Component Reliability Importance

Static Reliability Importance

Once the reliability of a system has been determined, engineers are often faced with the task of identifying the least reliable component(s) in the system in order to improve the design. For example, it was observed in RBDs and Analytical System Reliability that the least reliable component in a series system has the biggest effect on the system reliability. In this case, if the reliability of the system is to be improved, then the efforts can best be concentrated on improving the reliability of that component first. In simple systems such as a series system, it is easy to identify the weak components. However, in more complex systems this becomes quite a difficult task. For complex systems, the analyst needs a mathematical approach that will provide the means of identifying and quantifying the importance of each component in the system.

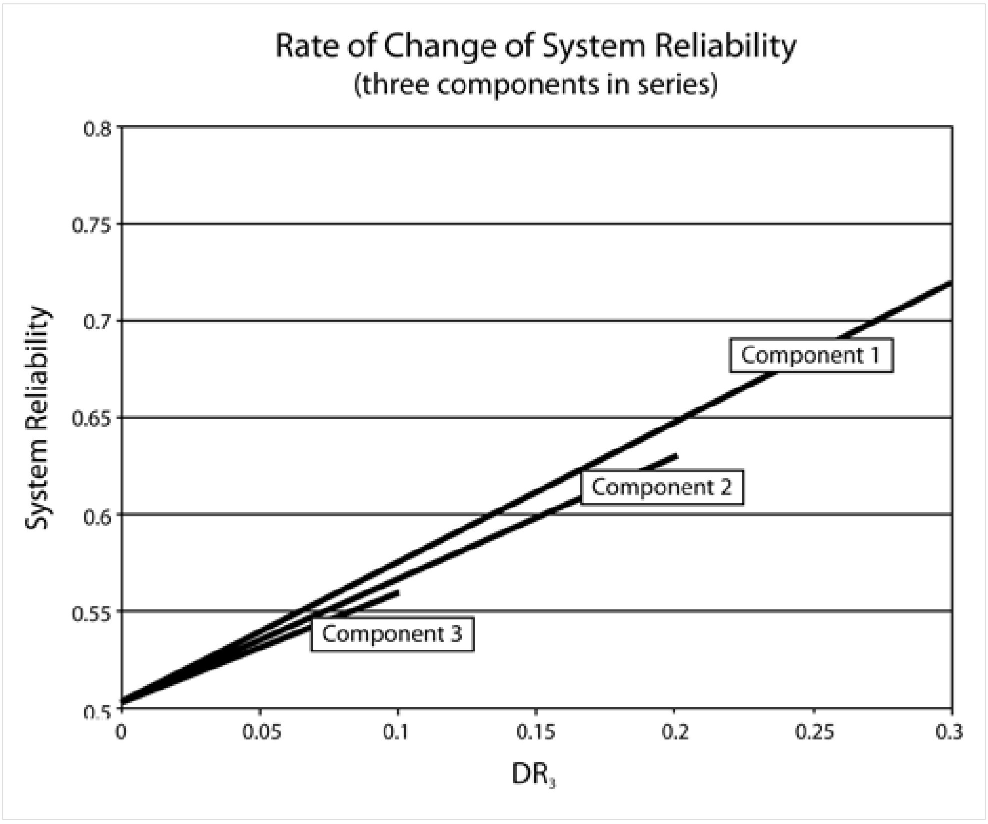
Using reliability importance measures is one method of identifying the relative importance of each component in a system with respect to the overall reliability of the system. The reliability importance, I_R, of component $i$in a system of $n$components is given by Leemis [17]:

$$I_{R_i} = \frac{\partial R_s}{\partial R_i}$$

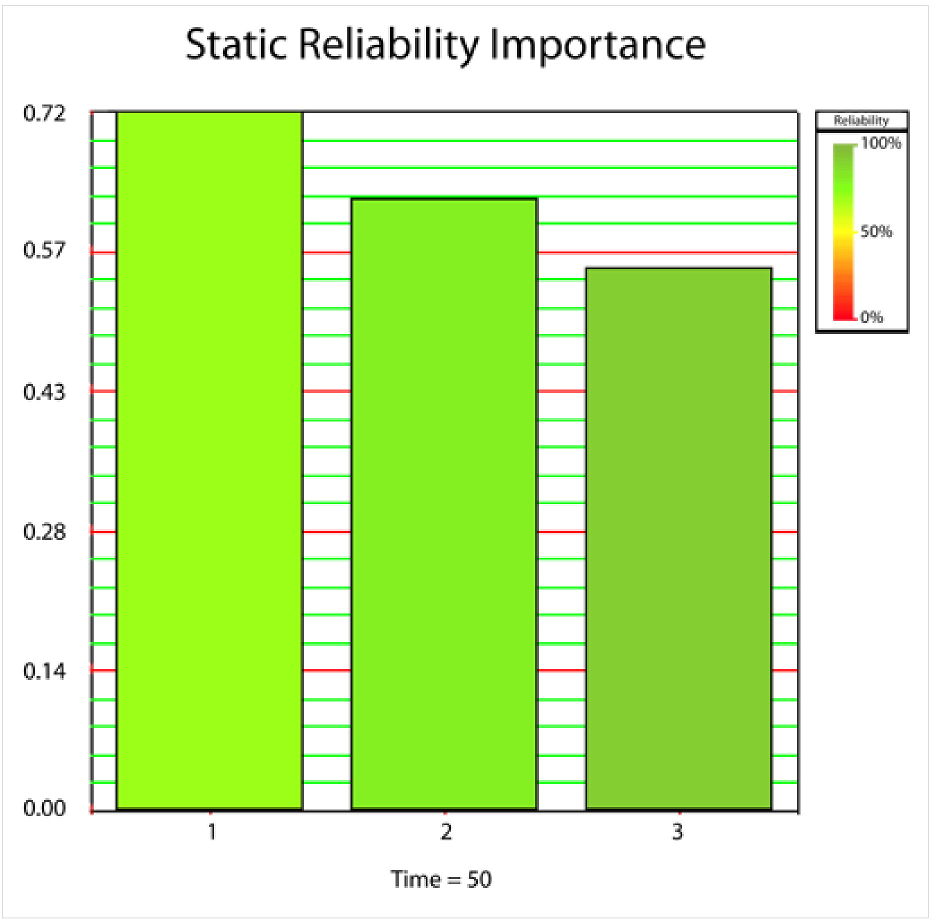
where:

- $R_s$is the system reliability.
- $R_i$is the component reliability.

The value of the reliability importance given by equation above depends both on the reliability of a component and its corresponding position in the system. In RBDs and Analytical System Reliability we observed that for a simple series system (three components in series with reliabilities of 0.7, 0.8 and 0.9) the rate of increase of the system reliability was greatest when the least reliable component was improved. In other words, it was observed that Component 1 had the largest reliability importance in the system relative to the other two components (see the figure below). The same conclusion can be drawn by using equation above and obtaining the reliability importance in terms of a value for each component.



Using BlockSim, the reliability importance values for these components can be calculated with the equation above. Using the plot option and selecting the Static Reliability Importance plot type, the following chart can be obtained. Note that the time input required to create this plot is irrelevant for this example because the components are static.



The values shown for each component were obtained using the equation above. The reliability equation for this series system is given by:

$$R_S = R_1 \cdot R_2 \cdot R_3$$

Taking the partial derivative of the equation above with respect to R_1yields:

$$\begin{aligned} I_{R_1} = \frac{\partial R_s}{\partial R_1} &= R_2 R_3 \\ &= 0.8 \cdot 0.9 \\ &= 0.72 \end{aligned}$$

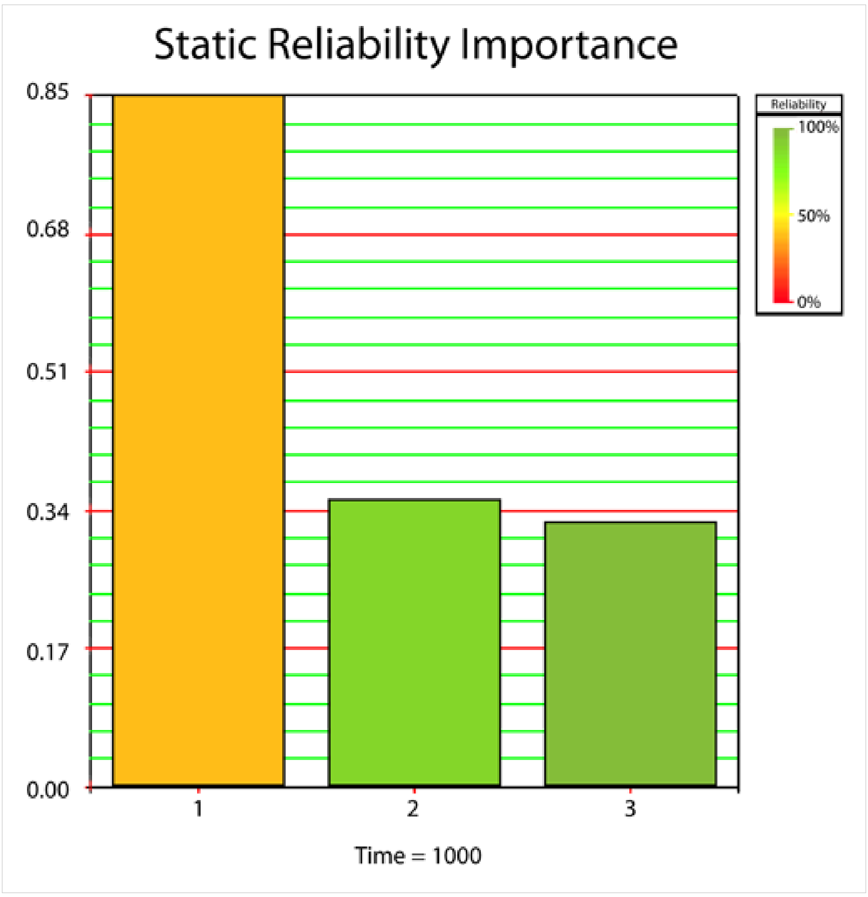
Thus the reliability importance of Component 1 is $I_{R_1} =$0.72. The reliability importance values for Components 2 and 3 are obtained in a similar manner.

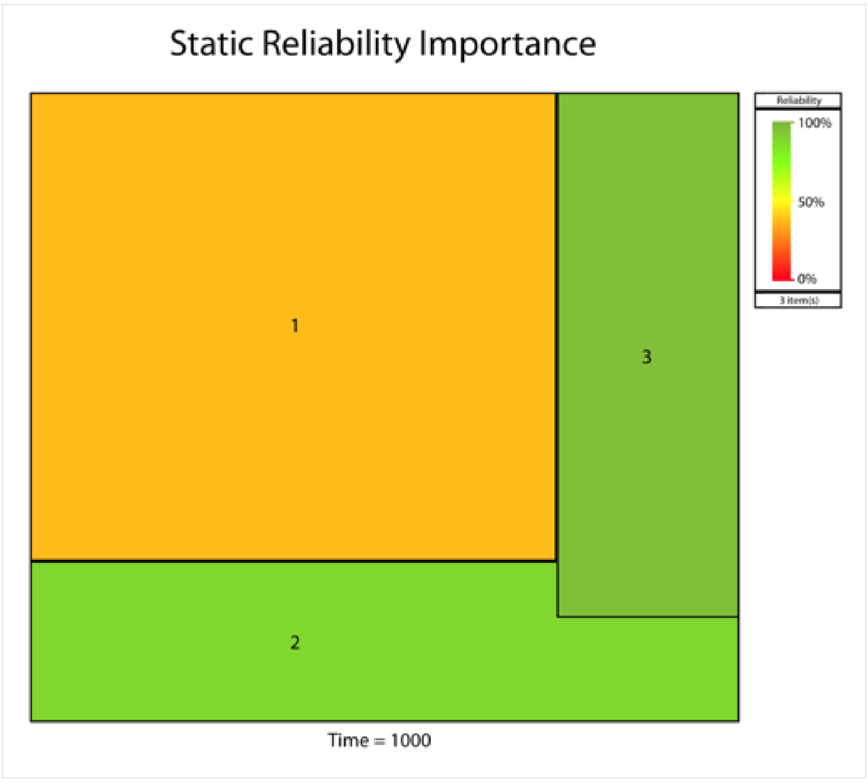
Time-Dependent Reliability Importance

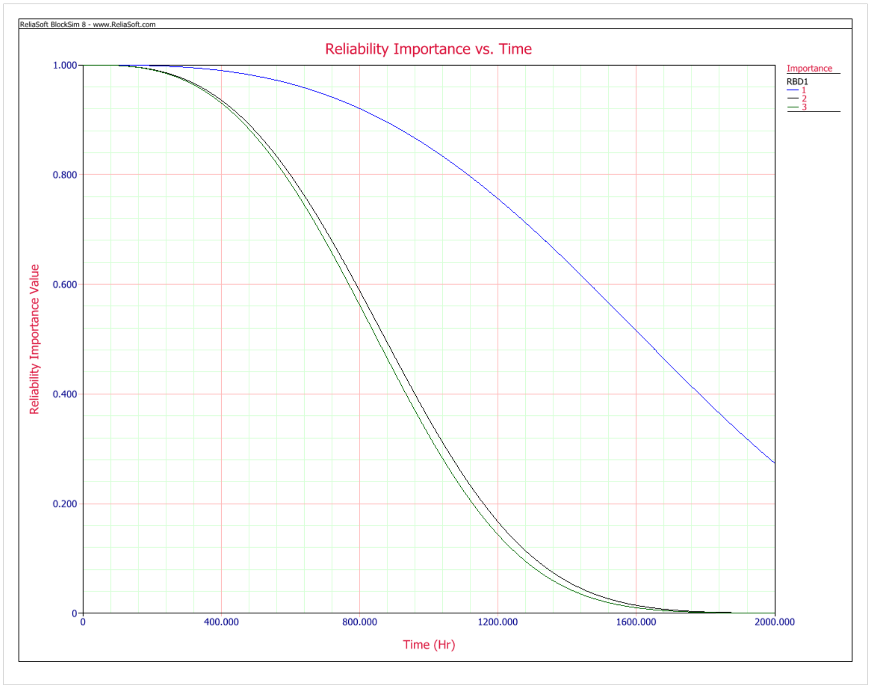
The same concept applies if the components have a time-varying reliability. That is, if $R_s(t) = R_1(t) \cdot R_2(t) \cdot R_3(t)$, then one could compute I_{R_i}at any time x, or $I_{R_i}(t)_{t=x}$. This is quantified in the following equation.

$$I_{R_i}(t) = \frac{\partial R_s(t)}{\partial R_i(t)}$$

In turn, this can be viewed as either a static plot (at a given time) or as a time-varying plot, as illustrated in the next figures. Specifically, the next three plots present the analysis for three components configured reliability-wise in series following a Weibull distribution with $\beta = 3$and $\eta_1 = 1{,}000$, $\eta_2 = 2{,}000$and $\eta_3 = 3{,}000$. The first shows a bar chart of I_{R_i}while the second shows the I_{R_i}in BlockSim's tableau chart format. In this chart, the area of the square is I_{R_i}. Finally, the third plot shows the $I_{R_i}(t)$vs. time.


Static Reliability Importance
0.85
0.68
0.51
0.34
0.17
0.00
1
2
3
Time = 1000
Reliability
100%
50%
0%


Static Reliability Importance
1
2
3
Time = 1000
Reliability
100%
50%
0%
3 item(s)



Example

Reliability Importance Measures for Failure Modes

Assume that a system has failure modes A, B, C, D, $E$and F. Furthermore, assume that failure of the entire system will occur if:

- Mode B, Cor Foccurs.
- Modes $A$and E, $A$and Dor $E$and Doccur.

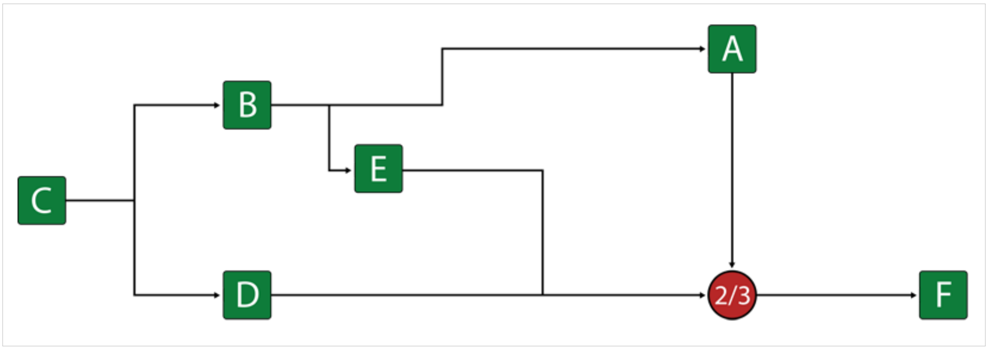
In addition, assume the following failure probabilities for each mode.

- Modes $A$and Dhave a mean time to occurrence of 1,000 hours (i.e., exponential with $MTTF = 1,000)$.
- Mode Ehas a mean time to occurrence of 100 hours (i.e., exponential with $MTTF = 100)$.
- Modes $B, C$and Fhave a mean time to occurrence of 700,000, 1,000,000 and 2,000,000 hours respectively (i.e., exponential with $MTTF_B = 700,000$, $MTTF_C = 1,000,000$and $MTTF_F = 2,000,000)$.

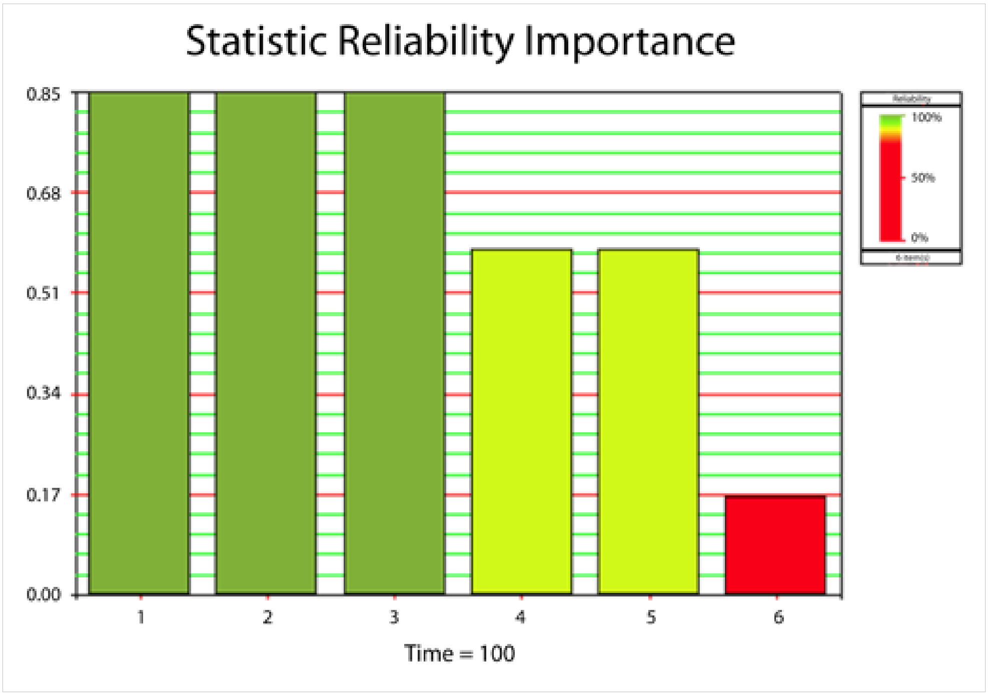
Examine the mode importance for operating times of 100 and 500 hours.

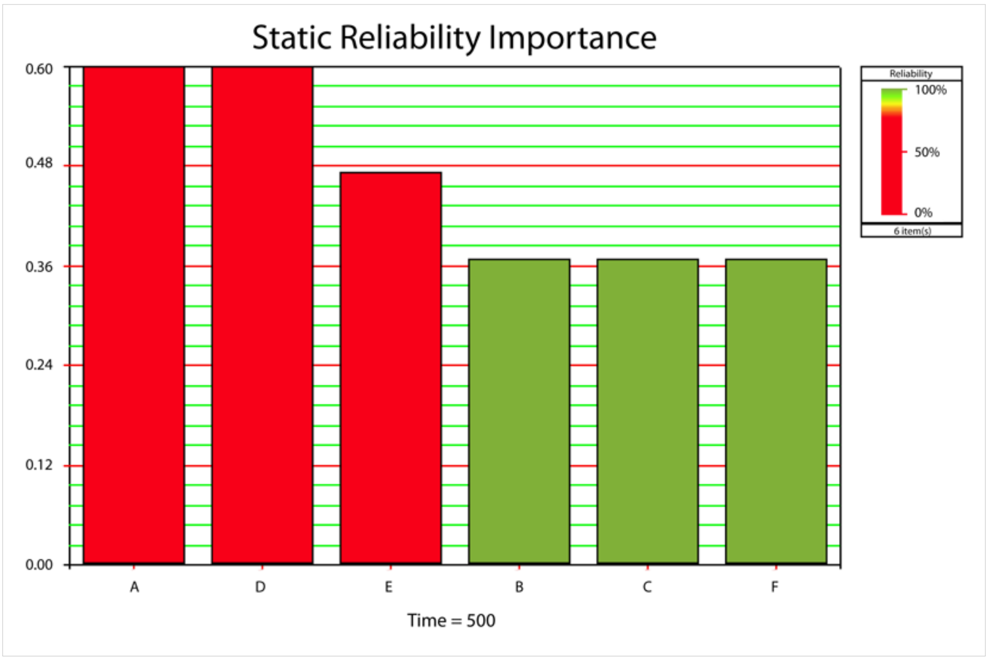
Solution

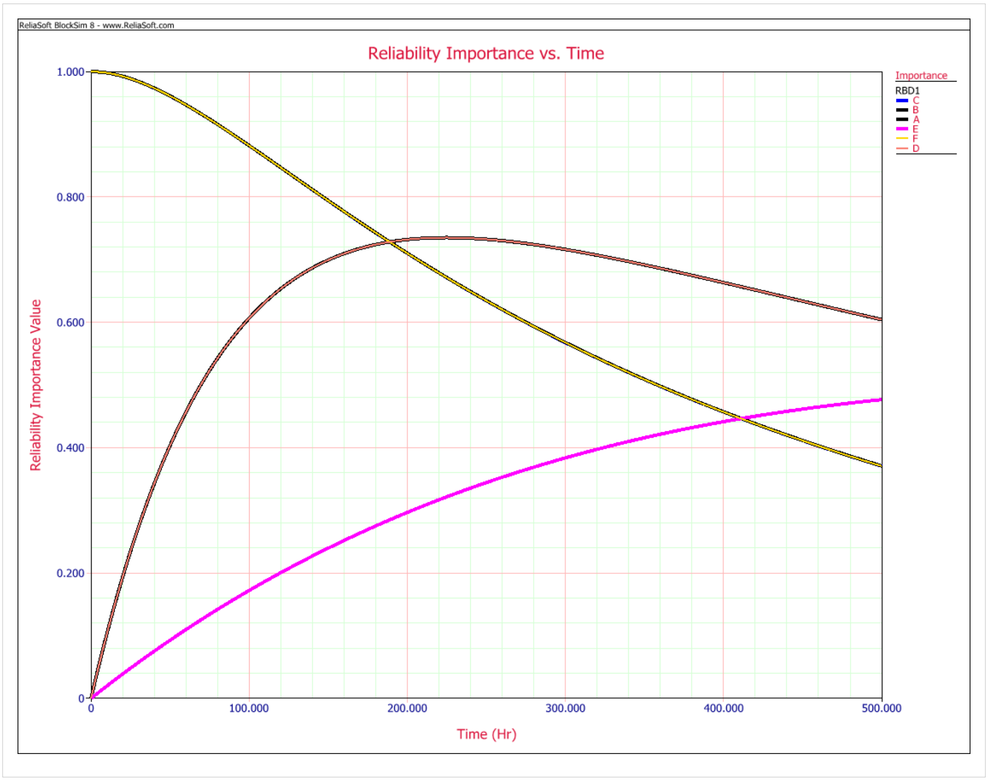
The RBD for this example is shown next:



The first chart below illustrates $I_{R_i}(t = 100)$. It can be seen that even though $B, C$and Fhave a much rarer rate of occurrence, they are much more significant at 100 hours. By 500 hours, $I_{R_i}(t = 500)$, the effects of the lower reliability components become greatly pronounced and thus they become more important, as can be seen in the second chart. Finally, the behavior of $I_{R_i}(t)$can be observed in the Reliability Importance vs. Time plot. Note that not all lines are plainly visible in the plot due to overlap.







Importance Measures and FMEA/FMECA

Traditional Failure Mode and Effects Analysis (FMEA/FMECA) relies on Risk Priority Numbers (RPNs) or criticality calculations to identify and prioritize the significance/importance of different failure modes. The RPN methodology (and to some extent, the criticality methodology) tend to be subjective. When conducting these types of analyses, one may wish to incorporate more quantitative metrics, such as the importance measures presented here and/or the RS FCI and RS DECI for repairable systems (which are discussed in later chapters). ReliaSoft's software [1] can be used to export an FMEA/FMECA analysis to BlockSim. The documentation that accompanies Xfmea provides more information on FMEA/FMECA, including both methods of risk assessment.

Reliability Allocation

In the process of developing a new product, the engineer is often faced with the task of designing a system that conforms to a set of reliability specifications. The engineer is given the goal for the system and must then develop a design that will achieve the desired reliability of the system, while performing all of the system's intended functions at a minimum cost. This involves a balancing act of determining how to allocate reliability to the components in the system so the system will meet its reliability goal while at the same time ensuring that the system meets all of the other associated performance specifications.

BlockSim provide three allocation methods: equal allocation, weighted reliability allocation and cost optimzation allocation. In these three methods, the simplest method is equal reliability allocation, which distributes the reliabilities uniformly among all components. For example, suppose a system with five components in series has a reliability objective of 90% for a given operating time. The uniform allocation of the objective to all components would require each component to have a reliability of 98% for the specified operating time, since $0.98^5 \simeq 0.90$. While this manner of allocation is easy to calculate, it is generally not the best way to allocate reliability for a system. The optimum method of allocating reliability would take into account the cost or relative difficulty of improving the reliability of different subsystems or components.

The reliability optimization process begins with the development of a model that represents the entire system. This is accomplished with the construction of a system reliability block diagram that represents the reliability relationships of the components in the system. From this model, the system reliability impact of different component modifications can be estimated and considered alongside the costs that would be incurred in the process of making those modifications. It is then possible to perform an optimization analysis for this problem, finding the best combination of component reliability improvements that meet or exceed the performance goals at the lowest cost.

Improving Reliability

Reliability engineers are very often called upon to make decisions as to whether to improve a certain component or components in order to achieve a minimum required system reliability. There are two approaches to improving the reliability of a system: fault avoidance and fault tolerance. Fault avoidance is achieved by using high-quality and high-reliability components and is usually less expensive than fault tolerance. Fault tolerance, on the other hand, is achieved by redundancy. Redundancy can result in increased design complexity and increased costs through additional weight, space, etc.

Before deciding whether to improve the reliability of a system by fault tolerance or fault avoidance, a reliability assessment for each component in the system should be made. Once the reliability values for the components have been quantified, an analysis can be performed in order to determine if that system's reliability goal will be met. If it becomes apparent that the system's reliability will not be adequate to meet the desired goal at the specified mission duration, steps can be taken to determine the best way to improve the system's reliability so that it will reach the desired target.

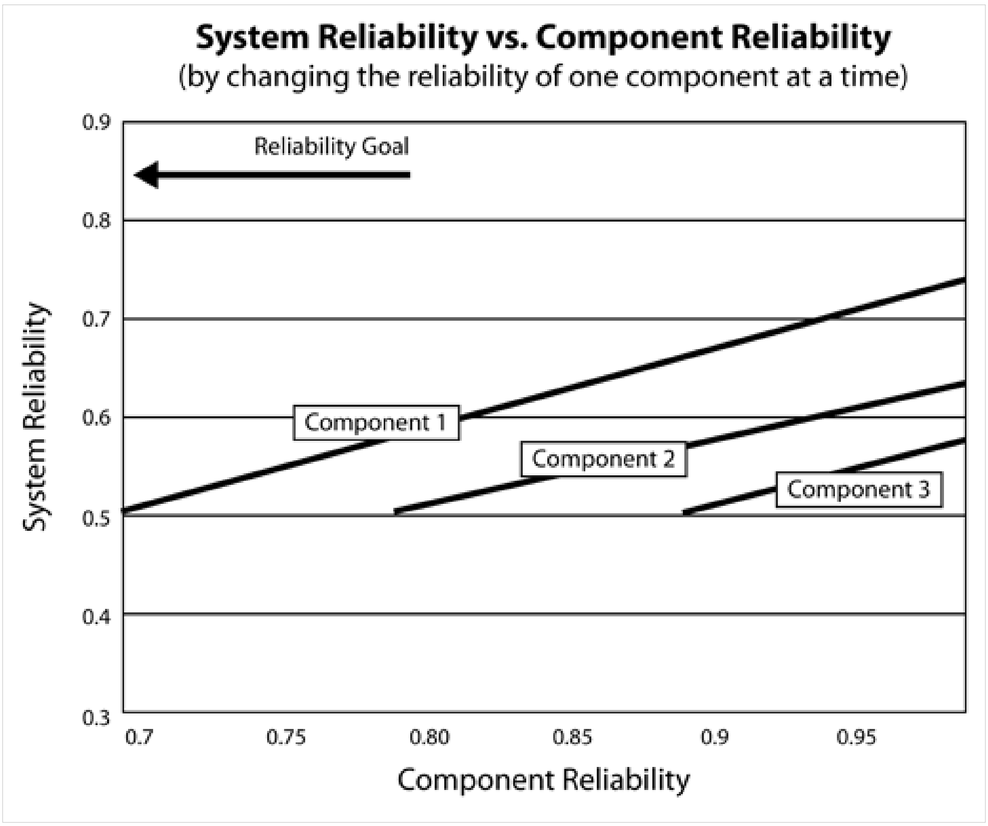
Consider a system with three components connected reliability-wise in series. The reliabilities for each component for a given time are: $R_1 = 70\%, R_2 = 80\%$, and $R_3 = 90\%$. A reliability goal, $R_G = 85\%$is required for this system.

The current reliability of the system is:

$$R_s = R_1 \cdot R_2 \cdot R_3 = 50.4\%$$

Obviously, this is far short of the system's required reliability performance. It is apparent that the reliability of the system's constituent components will need to be increased in order for the system to meet its goal. First, we will try increasing the reliability of one component at a time to see whether the reliability goal can be achieved.

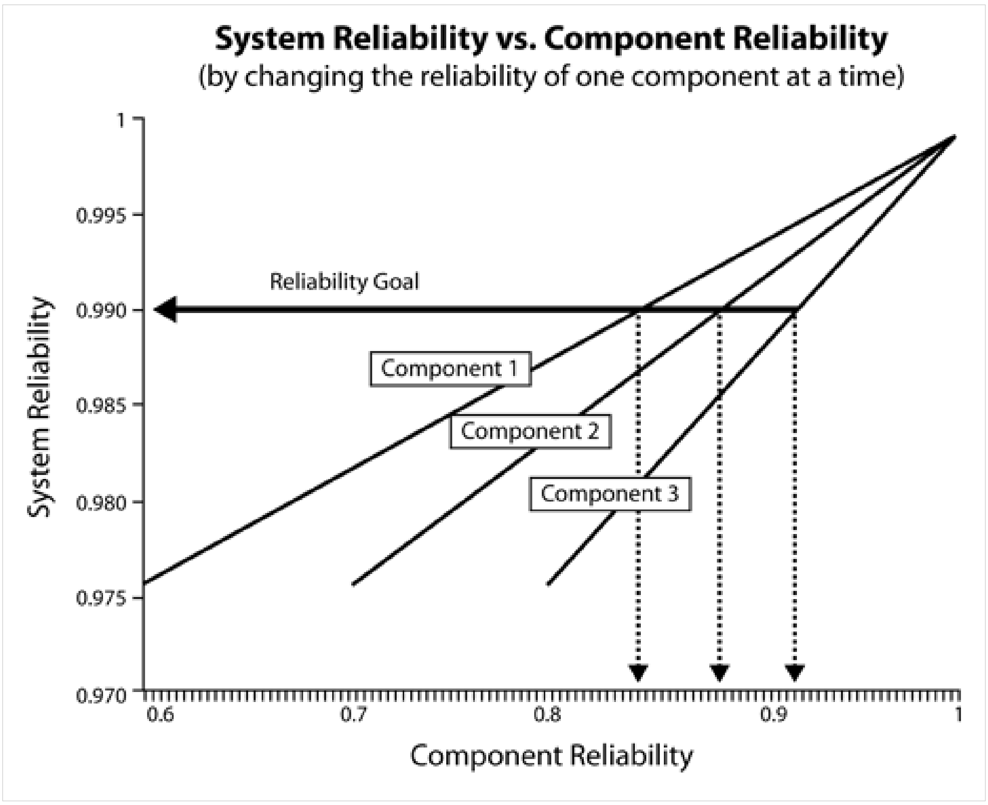
The following figure shows that even by raising the individual component reliability to a hypothetical value of 1 (100% reliability, which implies that the component will never fail), the overall system reliability goal will not be met by improving the reliability of just one component. The next logical step would be to try to increase the reliability of two components. The question now becomes: which two? One might also suggest increasing the reliability of all three components. A basis for making such decisions needs to be found in order to avoid the *trial and error* aspect of altering the system's components randomly in an attempt to achieve the system reliability goal.



As we have seen, the reliability goal for the preceding example could not be achieved by increasing the reliability of just one component. There are cases, however, where increasing the reliability of one component results in achieving the system reliability goal. Consider, for example, a system with three components connected reliability-wise in parallel. The reliabilities for each component for a given time are: $R_1 = 60\%$, $R_2 = 70\%$and $R_3 = 80\%$. A reliability goal, $R_G = 99\%$,is required for this system. The initial system reliability is:

$$R_S = 1 - (1 - 0.6) \cdot (1 - 0.7) \cdot (1 - 0.8) = 0.976$$

The current system reliability is inadequate to meet the goal. Once again, we can try to meet the system reliability goal by raising the reliability of just one of the three components in the system.



From the above figure, it can be seen that the reliability goal can be reached by improving Component 1, Component 2 or Component 3. The reliability engineer is now faced with another dilemma: which component's reliability should be improved? This presents a new aspect to the problem of allocating the reliability of the system. Since we know that the system reliability goal can be achieved by increasing at least one unit, the question becomes one of how to do this most efficiently and cost effectively. We will need more information to make an informed decision as to how to go about improving the system's reliability. How much does each component need to be improved for the system to meet its goal? How feasible is it to improve the reliability of each component? Would it actually be more efficient to slightly raise the reliability of two or three components rather than radically improving only one?

In order to answer these questions, we must introduce another variable into the problem: cost. Cost does not necessarily have to be in dollars. It could be described in terms of non-monetary resources, such as time. By associating cost values to the reliabilities of the system's components, we can find an optimum design that will provide the required reliability at a minimum cost.

Cost/Penalty Function

There is always a cost associated with changing a design due to change of vendors, use of higher-quality materials, retooling costs, administrative fees, etc. The cost as a function of the reliability for each component must be quantified before attempting to improve the reliability. Otherwise, the design changes may result in a system that is needlessly expensive or overdesigned. Developing the "cost of reliability" relationship will give the engineer an understanding of which components to improve and how to best concentrate the effort and allocate resources in doing so. The first step will be to obtain a relationship between the cost of improvement and reliability.

The preferred approach would be to formulate the cost function from actual cost data. This can be done from past experience. If a reliability growth program is in place, the costs associated with each stage of improvement can also be quantified. Defining the different costs associated with different vendors or different component models is also useful in formulating a model of component cost as a function of reliability.

However, there are many cases where no such information is available. For this reason, a general (default) behavior model of the cost versus the component's reliability was developed for performing reliability optimization in BlockSim. The objective of this function is to model an overall cost behavior for all types of components. Of course, it is impossible to formulate a model that will be precisely applicable to every situation; but the proposed relationship is general enough to cover most applications. In addition to the default model formulation, BlockSim does allow the definition of user-defined cost models.

Quantifying the Cost/Penalty Function

One needs to quantify a cost function for each component, C_i, in terms of the reliability, R_i, of each component, or:

$$C_i = f(R_i)$$

This function should:

- Look at the current reliability of the component, $R_{Current}$.
- Look at the maximum possible reliability of the component, R_{Max}.
- Allow for different levels of difficulty (or cost) in increasing the reliability of each component. It can take into account:
  - design issues.
  - supplier issues.
  - state of technology.
  - time-to-market issues, etc.

Thus, for the cost function to comply with these needs, the following conditions should be adhered to:

- The function should be constrained by the minimum and maximum reliabilities of each component (i.e., reliability must be less than one and greater than the current reliability of the component or at least greater than zero).
- The function should not be linear, but rather quantify the fact that it is incrementally harder to improve reliability. For example, it is considerably easier to increase the reliability from 90% to 91% than to increase it from 99.99% to 99.999%, even though the increase is larger in the first case.
- The function should be asymptotic to the maximum achievable reliability.

The following default cost function (also used in BlockSim) adheres to all of these conditions and acts like a penalty function for increasing a component's reliability. Furthermore, an exponential behavior for the cost is assumed since it should get exponentially more difficult to increase the reliability. See Mettas [21].

$$C_i(R_i) = e^{(1-f)\cdot \frac{R_i - R_{\min,i}}{R_{\max,i} - R_i}}$$

where:

- $C_i(R_i)$is the penalty (or cost) function as a function of component reliability.
- $f$is the feasibility (or cost index) of improving a component's reliability relative to the other components in the system.
- $R_{min,i}$is the current reliability at the time at which the optimization is to be performed.
- $R_{max,i}$is the maximum achievable reliability at the time at which the optimization is to be performed.

Note that this penalty function is dimensionless. It essentially acts as a weighting factor that describes the difficulty in increasing the component reliability from its current value, relative to the other components.

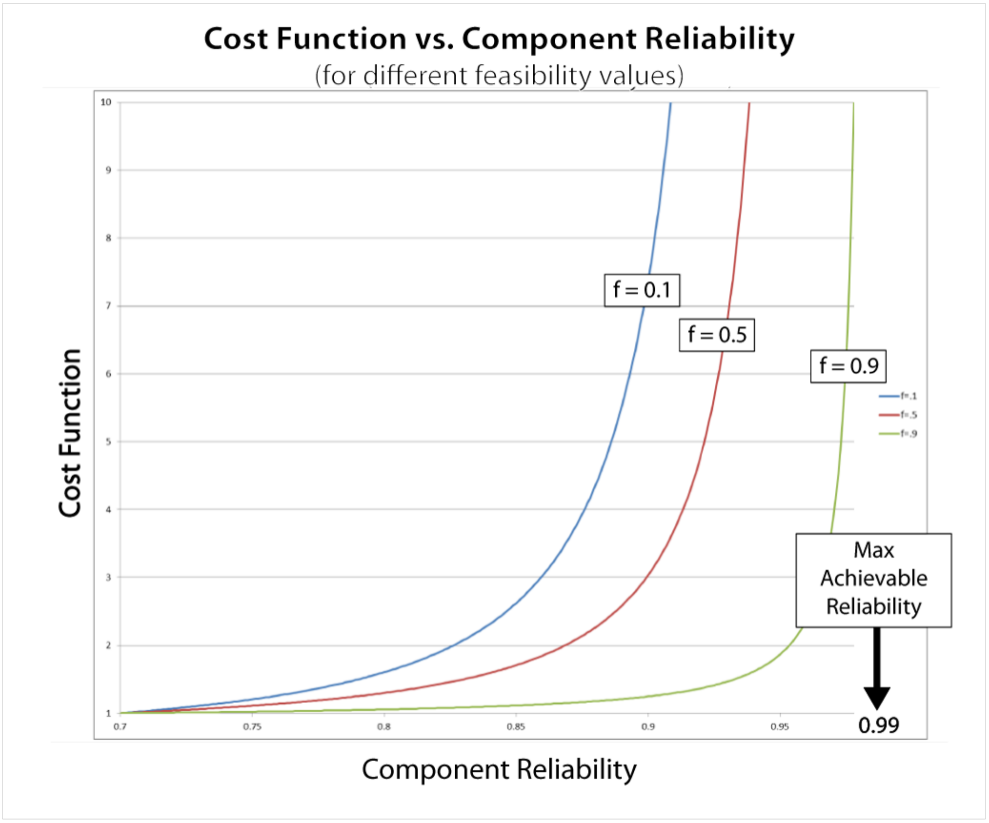
Examining the cost function given by equation above, the following observations can be made:

- The cost increases as the allocated reliability departs from the minimum or current value of reliability. It is assumed that the reliabilities for the components will not take values any lower than they already have. Depending on the optimization, a component's reliability may not need to be increased from its current value but it will not drop any lower.

- The cost increases as the allocated reliability approaches the maximum achievable reliability. This is a reliability value that is approached asymptotically as the cost increases but is never actually reached.
- The cost is a function of the range of improvement, which is the difference between the component's initial reliability and the corresponding maximum achievable reliability.
- The exponent in the above equation approaches infinity as the component's reliability approaches its maximum achievable value. This means that it is easier to increase the reliability of a component from a lower initial value. For example, it is easier to increase a component's reliability from 70% to 75% than to increase its reliability from 90% to 95%.

The Feasibility Term, f

The feasibility term in $C_i(R_i) = e^{(1-f)\cdot\frac{R_i - R_{\min,i}}{R_{\max,i} - R_i}}$is a constant (or an equation parameter) that represents the difficulty in increasing a component's reliability relative to the rest of the components in the system. Depending on the design complexity, technological limitations, etc., certain components can be very hard to improve. Clearly, the more difficult it is to improve the reliability of the component, the greater the cost. The following figure illustrates the behavior of the function defined in the above equation for different values of f. It can be seen that the lower the feasibility value, the more rapidly the cost function approaches infinity.



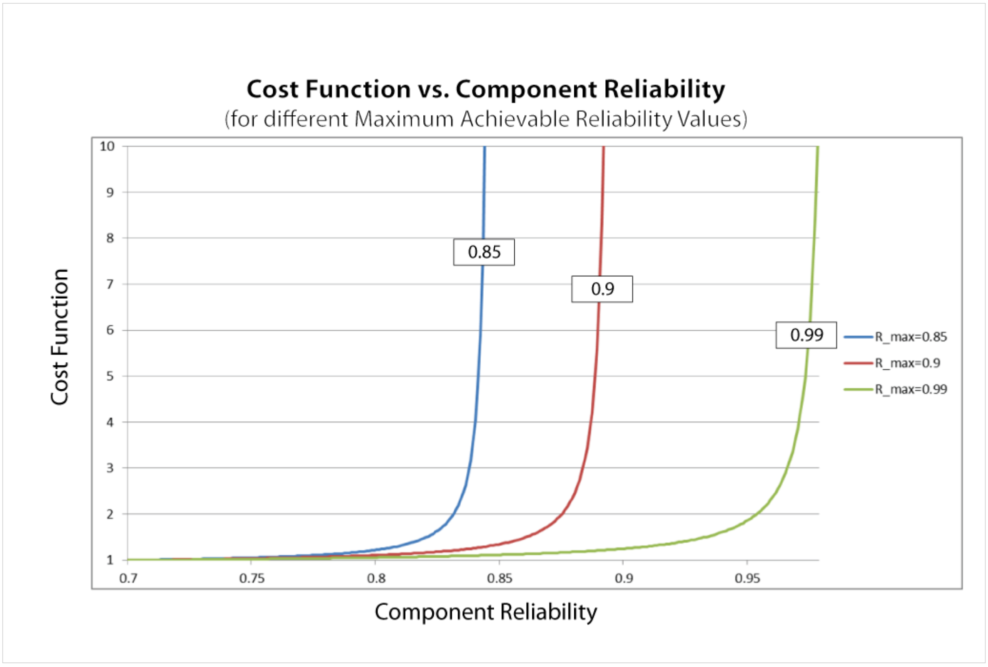
Several methods can be used to obtain a feasibility value. Weighting factors for allocating reliability have been proposed by many authors and can be used to quantify feasibility. These weights depend on certain factors of influence, such as the complexity of the component, the state of the art, the operational profile, the criticality, etc. Engineering judgment based on past experience, supplier quality, supplier availability and other factors can also be used in determining a feasibility value. Overall, the assignment of a feasibility value is going to be a subjective process. Of course, this problem is negated if the relationship between the cost and the reliability for each component is known because one can use regression methods to estimate the parameter value.

Maximum Achievable Reliability

For the purposes of reliability optimization, we also need to define a limiting reliability that a component will approach, but not reach. The costs near the maximum achievable reliability are very high and the actual value for the maximum reliability is usually dictated by technological or financial constraints. In deciding on a value to use for the maximum achievable reliability, the current state of the art of the component in question and other similar factors will have to be considered. In the end, a realistic estimation based on engineering judgment and experience will be necessary to assign a value to this input.

Note that the time associated with this maximum achievable reliability is the same as that of the overall system reliability goal. Almost any component can achieve a very high reliability value, provided the mission time is short enough. For example, a component with an exponential distribution and a failure rate of one failure per hour has a reliability that drops below 1% for missions greater than five hours. However, it can achieve a reliability of 99.9% as long as the mission is no longer than four seconds. For the purposes of optimization in BlockSim, the reliability values of the components are associated with the time for which the system reliability goal is specified. For example, if the problem is to achieve a system goal of 99% reliability at 1,000 hours, the maximum achievable reliability values entered for the individual components would be the maximum reliability that each component could attain for a mission of 1,000 hours.

As the component reliability, R_i, approaches the maximum achievable reliability, $R_{i,max}$, the cost function approaches infinity. The maximum achievable reliability acts as a scale parameter for the cost function. By decreasing $R_{i,max}$, the cost function is compressed between $R_{i,min}$and $R_{i,max}$, as shown in the figure below.



Cost Function

Once the cost functions for the individual components have been determined, it becomes necessary to develop an expression for the overall system cost. This takes the form of:

$$C_s(R_G) = C_1(R_1) + C_2(R_2) + ... + C_n(R_n), i = 1, 2, ..., n$$

In other words, the cost of the system is simply the sum of the costs of its components. This is regardless of the form of the individual component cost functions. They can be of the general behavior model in BlockSim or they can be user-defined. Once the overall cost function for the system has been defined, the problem becomes one of minimizing the cost function while remaining within the constraints defined by the target system reliability and the reliability ranges for the components. The latter constraints in this case are defined by the minimum and maximum reliability values for the individual components.

BlockSim employs a nonlinear programming technique to minimize the system cost function. The system has a minimum (current) and theoretical maximum reliability value that is defined by the minimum and maximum reliabilities of the components and by the way the system is configured. That is, the structural properties of the system are accounted for in the determination of the optimum solution. For example, the optimization for a system of three units in series will be different from the optimization for a system consisting of the same three units in parallel. The optimization occurs by varying the reliability values of the components within their respective constraints of maximum and minimum reliability in a way that the overall system goal is achieved. Obviously, there can be any number of different combinations of component reliability values that might achieve the system goal. The optimization routine essentially finds the combination that results in the lowest overall system cost.

Determining the Optimum Allocation Scheme

To determine the optimum reliability allocation, the analyst first determines the system reliability equation (the objective function). As an example, and again for a trivial system with three components in series, this would be:

$$R_S = R_1 \cdot R_2 \cdot R_3$$

If a target reliability of 90% is sought, then the equation above is recast as:

$$0.90 = R_1 \cdot R_2 \cdot R_3$$

The objective now is to solve for R_1, $R_2$and R_3so that the equality in the equation above is satisfied. To obtain an optimum solution, we also need to use our cost functions (i.e., define the total allocation costs) as:

$$C_T = C_1(R_1) + C_2(R_2) + C_3(R_3)$$

With the cost equation defined, then the optimum values for R_1, $R_2$and $R_3$are the values that satisfy the reliability requirement, the second equation above, at the minimum cost, the last equation above. BlockSim uses this methodology during the optimization task.

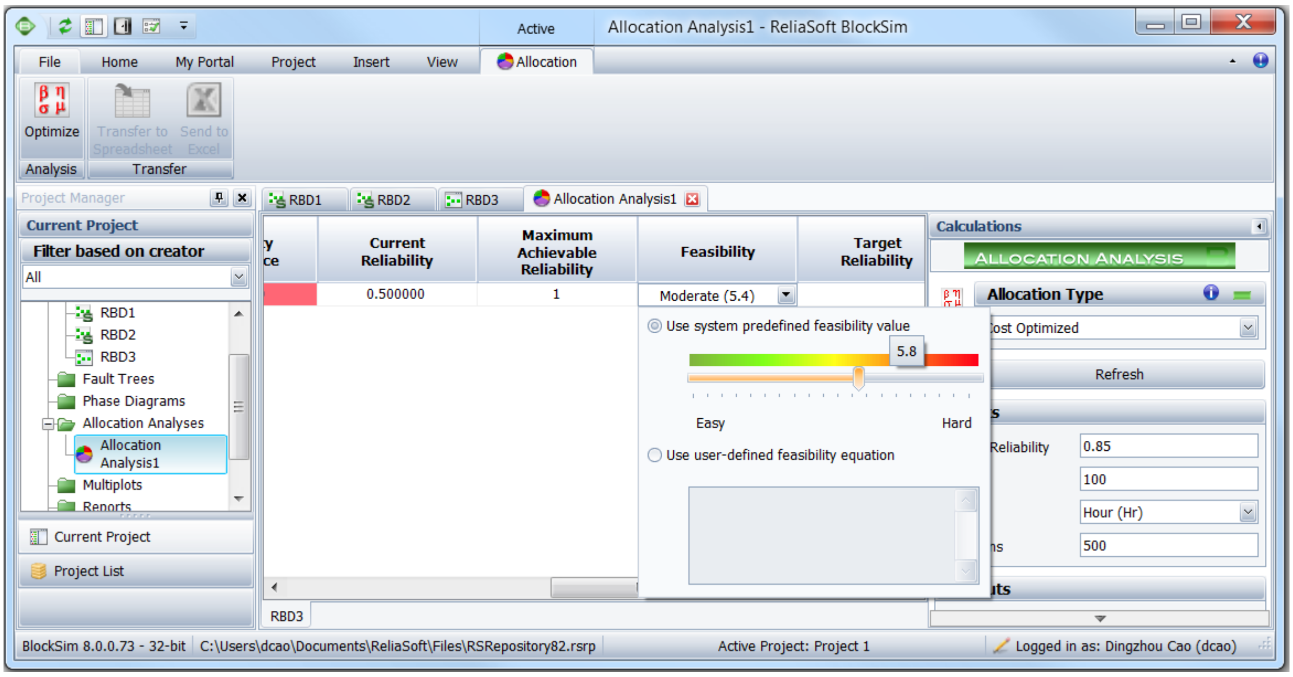
Defining a Feasibility Policy in BlockSim

In BlockSim, you can choose to use the default feasibility function, as defined by $C_i(R_i) = e^{(1-f)\cdot\frac{R_i - R_{\min,i}}{R_{\max,i} - R_i}}$, or use your own function. The first picture below illustrates the use of the default values using the slider control. The second figure shows the use of an associated feasibility policy to create a user-defined cost function. When defining your own cost function, you should be aware of/adhere to the following guidelines:

- Because the cost functions are evaluated relative to each other, they should be correlated. In other words, if one function evaluates to 10, $C_i(R_i) = 10$for one block and 20 for another, $C_i(R_i) = 20$, then the implication is that there is a 1 to 2 cost relation.
- Do not mix your own function with the software's default functions unless you have verified that your cost functions are defined and correlated to the default cost functions, as defined by:

$$C_i(R_i) = e^{(1-f)\cdot\frac{R_i - R_{\min,i}}{R_{\max,i} - R_i}}$$

- Your function should adhere to the guidelines presented earlier.
- Lastly, and since the evaluation is relative, it is preferable to use the predefined functions unless you have a compelling reason (or data) to do otherwise. The last section in this chapter describes cases where user-defined functions are preferred.



y ce	Current Reliability	Maximum Achievable Reliability	Feasibility	Target Reliability
	0.500000	1	Moderate (5.4)	

Use system predefined feasibility value

5.8

Use user-defined feasibility equation

Cost Optimized

Reliability 0.85

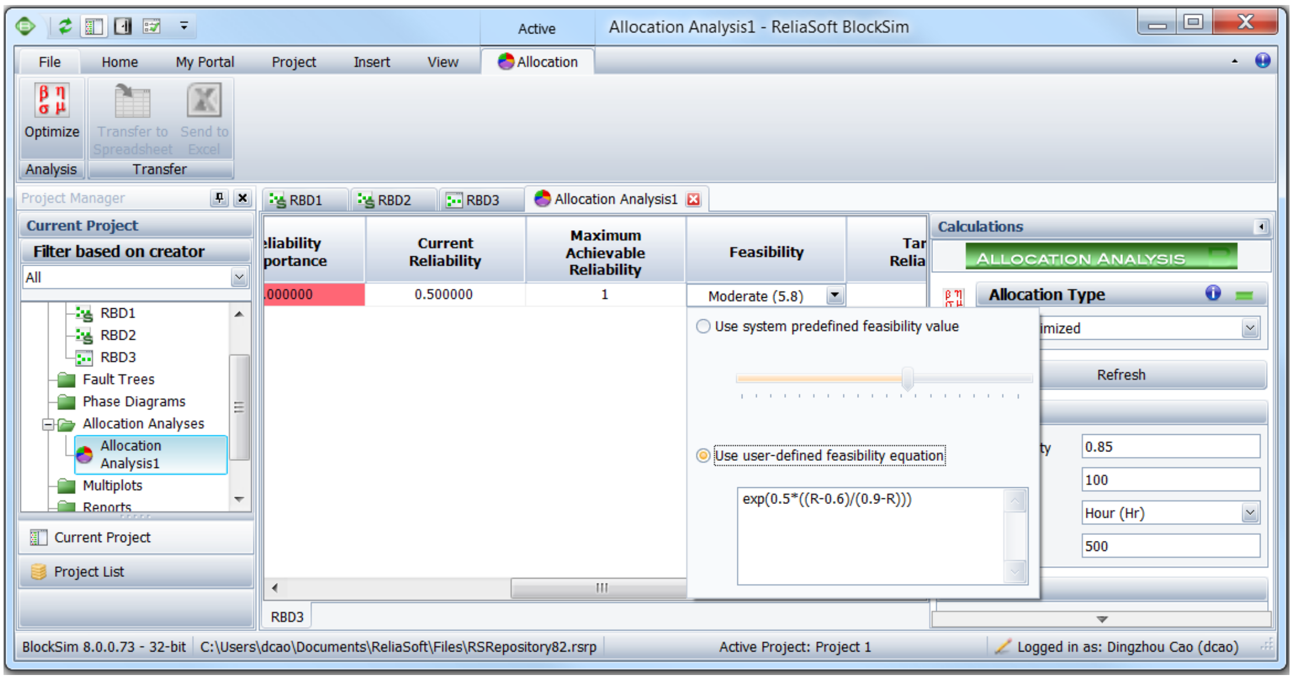
100

Hour (Hr)

500

BlockSim 8.0.0.73 - 32-bit C:\Users\dcao\Documents\ReliaSoft\Files\RSRepository82.rsrp Active Project: Project 1 Logged in as: Dingzhou Cao (dcao)

Setting the default feasibility function in BlockSim with the feasibility slider. Note that the feasibility slider displays values, *SV*, from 0.1 to 9.9 when moved by the user, with SV=9.9 being the hardest. The relationship between *f* and *SV* is "f = (1 - SV/10)").



eliability portance	Current Reliability	Maximum Achievable Reliability	Feasibility	Tar Relia
.000000	0.500000	1	Moderate (5.8)	

Use system predefined feasibility value

Use user-defined feasibility equation

exp(0.5*((R-0.6)/(0.9-R)))

imized

ty 0.85

100

Hour (Hr)

500

BlockSim 8.0.0.73 - 32-bit C:\Users\dcao\Documents\ReliaSoft\Files\RSRepository82.rsrp Active Project: Project 1 Logged in as: Dingzhou Cao (dcao)

Setting a user-defined feasibility function in BlockSim. Any user-defined equation can be entered as a function of *R*.

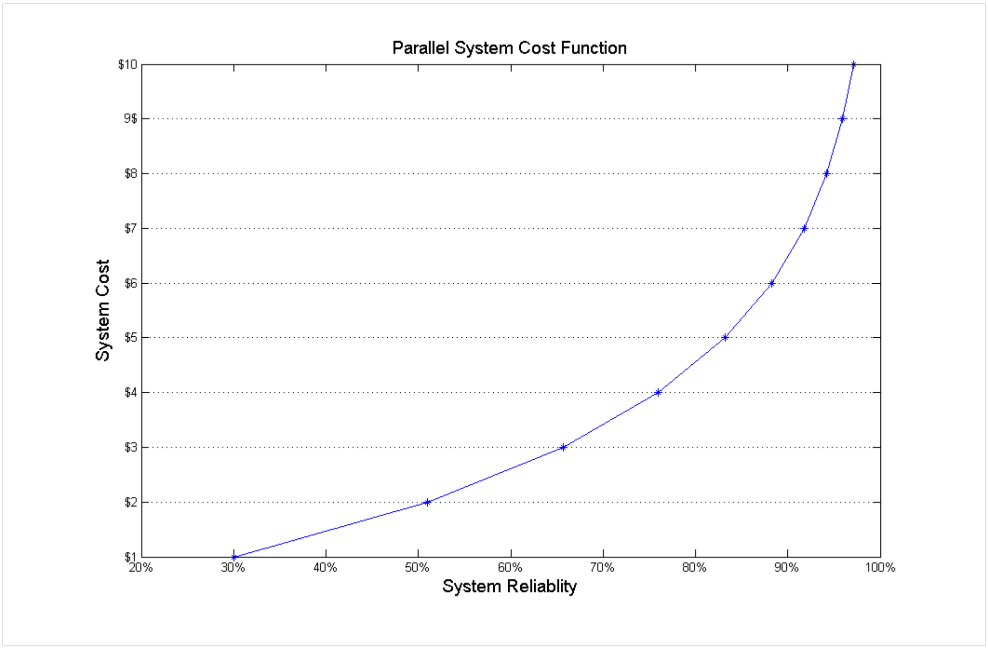
Implementing the Optimization

As was mentioned earlier, there are two different methods of implementing the changes suggested by the reliability optimization routine: fault tolerance and fault avoidance. When the optimized component reliabilities have been determined, it does not matter which of the two methods is employed to realize the optimum reliability for the component in question. For example, suppose we have determined that a component must have its reliability for a certain mission time raised from 50% to 75%. The engineer must now decide how to go about implementing the increase in reliability. If the engineer decides to do this via fault avoidance, another component must be found (or the existing component must be redesigned) so that it will perform the same function with a higher reliability. On the other hand, if the engineer decides to go the fault tolerance route, the optimized reliability can be achieved merely by placing a second identical component in parallel with the first one.

Obviously, the method of implementing the reliability optimization is going to be related to the cost function and this is something the reliability engineer must take into account when deciding on what type of cost function is used for the optimization. In fact, if we take a closer look at the fault tolerance scheme, we can see some parallels with the general behavior cost model included in BlockSim. For example, consider a system that consists of a single unit. The cost of that unit, including all associated mounting and hardware costs, is one dollar. The reliability of this unit for a given mission time is 30%. It has been determined that this is inadequate and that a second component is to be added in parallel to increase the reliability. Thus, the reliability for the two-unit parallel system is:

$$R_S = 1 - (1 - 0.3)^2 = 0.51 \text{ or } 51\%$$

So, the reliability has increased by a value of 21% and the cost has increased by one dollar. In a similar fashion, we can continue to add more units in parallel, thus increasing the reliability and the cost. We now have an array of reliability values and the associated costs that we can use to develop a cost function for this fault tolerance scheme. The next figure shows the relationship between cost and reliability for this example.



As can be seen, this looks quite similar to the general behavior cost model presented earlier. In fact, a standard regression analysis available in Weibull++ indicates that an exponential model fits this cost model quite well. The function is given by the following equation, where $C$is the cost in dollars and $R$is the fractional reliability value.

$$C(R) = 0.3756 \cdot e^{3.1972 \cdot R}$$

Reliability Allocation Examples

Consider a system consisting of three components connected reliability-wise in series. Assume the objective reliability for the system is 90% for a mission time of 100 hours. Five cases will be considered for the allocation problem. See Mettas [21].

- **Case 1** - All three components are identical with times-to-failure that are described by a Weibull distribution with $\beta = 1.318$and $\eta = 312$hours. All three components have the same feasibility value of Moderate (5).
- **Case 2** - Same as in Case 1, but Component 1 has a feasibility of Easy, Component 2 has a feasibility of Moderate and Component 3 has a feasibility of Hard.
- **Case 3** - Component 1 has 70% reliability, Component 2 has 80% reliability and Component 3 has 90% reliability, all for a mission duration of 100 hours. All three components have the same feasibility of Easy.
- **Case 4** - Component 1 has 70% reliability and Easy feasibility, Component 2 has 80% reliability and Moderate feasibility, and Component 3 has 90% reliability and Hard feasibility, all for a mission duration of 100 hours.
- **Case 5** - Component 1 has 70% reliability and Hard feasibility, Component 2 has 80% reliability and Easy feasibility and Component 3 has 90% reliability and Moderate feasibility, all for a mission duration of 100 hours.

In all cases, the maximum achievable reliability, $R_{max,i}$, for each component is 99.9% for a mission duration of 100 hours.

Case 1 - The reliability equation for Case 1 is:

$$R_S(t) = R_1(t) \cdot R_2(t) \cdot R_3(t)$$

Thus, the equality desired is:

$$0.90 = R_1(t=100) \cdot R_2(t=100) \cdot R_3(t=100)$$

where:

$$R_{1,2,3} = e^{-\left(\frac{t}{\eta}\right)^{\beta}}$$

The cost or feasibility function is:

$$C_T = C_1(R_1) + C_2(R_2) + C_3(R_3)$$

where:

$$C_{1,2,3}(R_{1,2,3}) = e^{(1-f)\cdot \frac{R_i - R_{\text{min},i}}{R_{\text{max},i} - R_i}}$$

And where $R_{\text{max}_{1,2,3}} = 0.999$(arbitrarily set), $R_{\text{min}_{1,2,3}}$computed from the reliability function of each component at the time of interest, $t = 100$, or:

$$\begin{aligned} R_{\text{min}_{1,2,3}} =& e^{-\left(\frac{t}{\eta}\right)^{\beta}} \\ =& e^{-\left(\frac{100}{312}\right)^{1.318}} \\ =& 0.79995 \end{aligned}$$

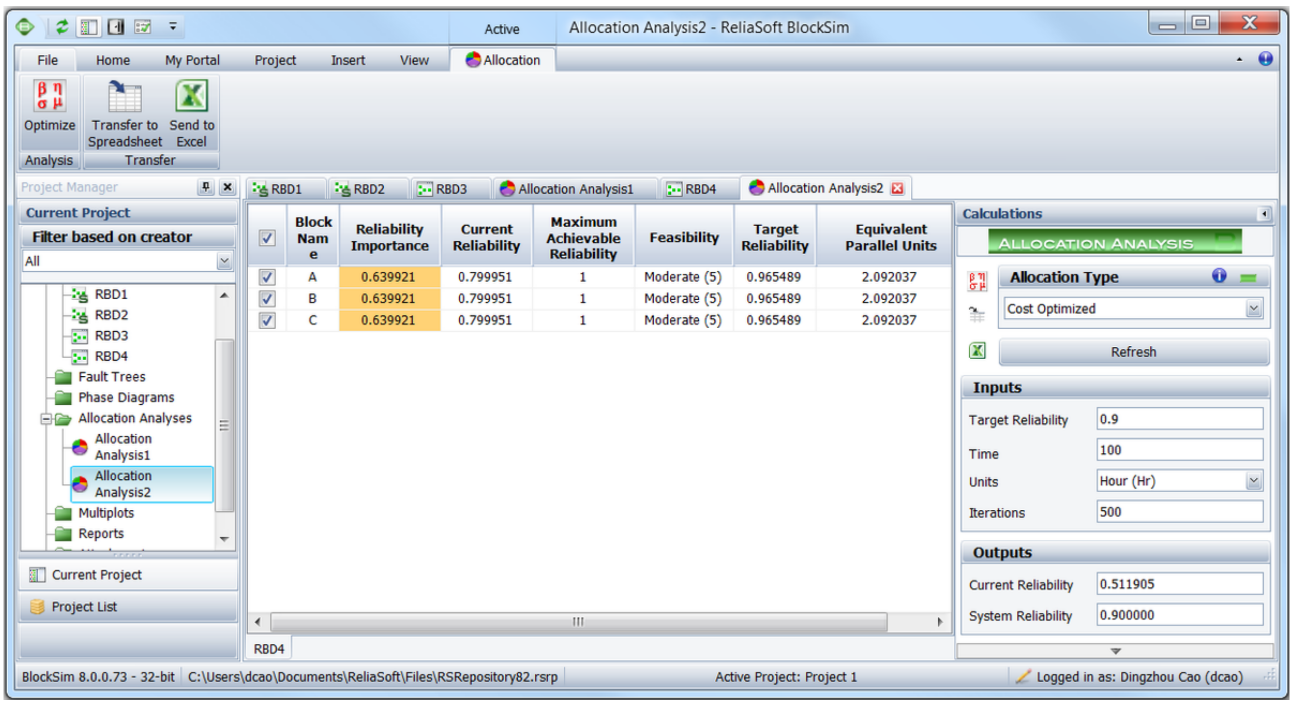
And $f$obtained from:

$$\begin{aligned} f =& \left(1 - \frac{5}{10}\right) \\ =& 0.5 \end{aligned}$$

The solution, R_{O_i}, is the one that satisfies $0.90 = R_1(t=100) \cdot R_2(t=100) \cdot R_3(t=100)$while minimizing $C_T = C_1(R_1) + C_2(R_2) + C_3(R_3)$. In this case (and since all the components are identical), the target reliability is found to be:

$$R_{O_i}(t=100) = 0.9655$$

The following figure shows the optimization results from BlockSim.



	Block Name	Reliability Importance	Current Reliability	Maximum Achievable Reliability	Feasibility	Target Reliability	Equivalent Parallel Units
✓	A	0.639921	0.799951	1	Moderate (5)	0.965489	2.092037
✓	B	0.639921	0.799951	1	Moderate (5)	0.965489	2.092037
✓	C	0.639921	0.799951	1	Moderate (5)	0.965489	2.092037

Allocation Type: Cost Optimized

Inputs

Target Reliability	0.9
Time	100
Units	Hour (Hr)
Iterations	500

Outputs

Current Reliability	0.511905
System Reliability	0.900000

BlockSim 8.0.0.73 - 32-bit C:\Users\dcao\Documents\ReliaSoft\Files\RSRepository82.rsrp Active Project: Project 1 Logged in as: Dingzhou Cao (dcao)

Based on this, each component's reliability should be at least 96.55% at 100 hours in order for the system's reliability to be 90% at 100 hours. Note the column labeled "Equivalent Parallel Units" which represents the number of redundant units that would be required to bring that particular component up to the recommended reliability. In the case where the fault tolerance approach is to be implemented, the "Equivalent Parallel Units" value should be rounded up to an integer. Therefore, some manipulation by the engineer is required in order to ensure that the chosen integer values will yield the required system reliability goal (or exceed it). In addition, further cost analysis should be performed in order to account for the costs of adding redundancy to the system.

Additionally, and when the results have been obtained, the engineer may wish to re-scale the components based on their distribution parameters instead of the fixed reliability value. In the case of these components, one may wish to re-scale the scale parameter of the distribution, η, for the components, or:

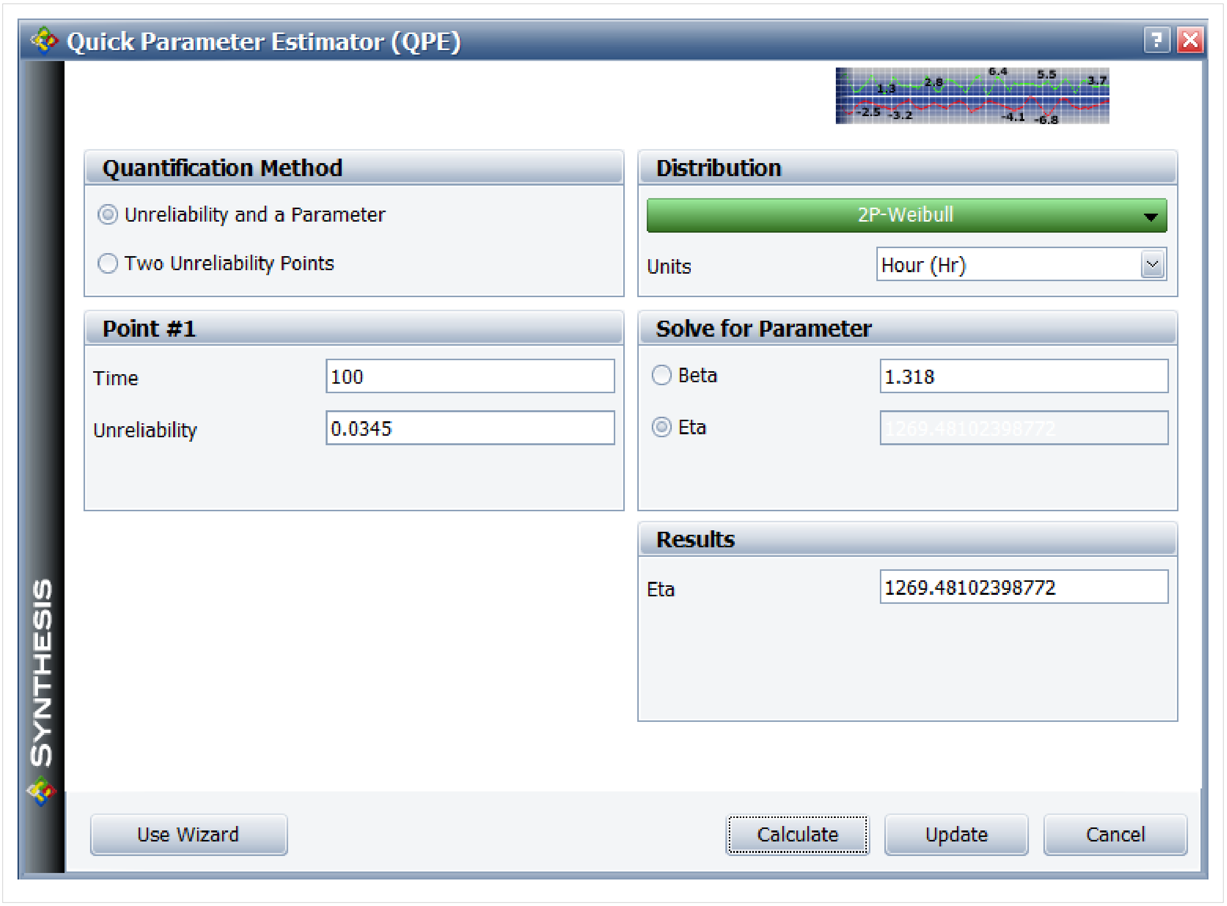
$$0.9655 = e^{-\left(\frac{t}{\eta}\right)^{\beta}}$$

$$0.9655 = e^{-\left(\frac{100}{\eta}\right)^{1.318}}$$

Which yields:

$$\eta_{O_i} = 1269.48$$

The Quick Parameter Estimator (QPE) in BlockSim can also be used for this, as shown below.



Quantification Method: [✓ Unreliability and a Parameter] Two Unreliability Points

Distribution: 2P-Weibull

Units: Hour (Hr)

Point #1

Time: 100

Unreliability: 0.0345

Solve for Parameter: Beta 1.318 [✓ Eta] 1269.48102398772

Results

Eta: 1269.48102398772

The results from the other cases can be obtained in a similar fashion. The results for Cases 1 through 5 are summarized next.

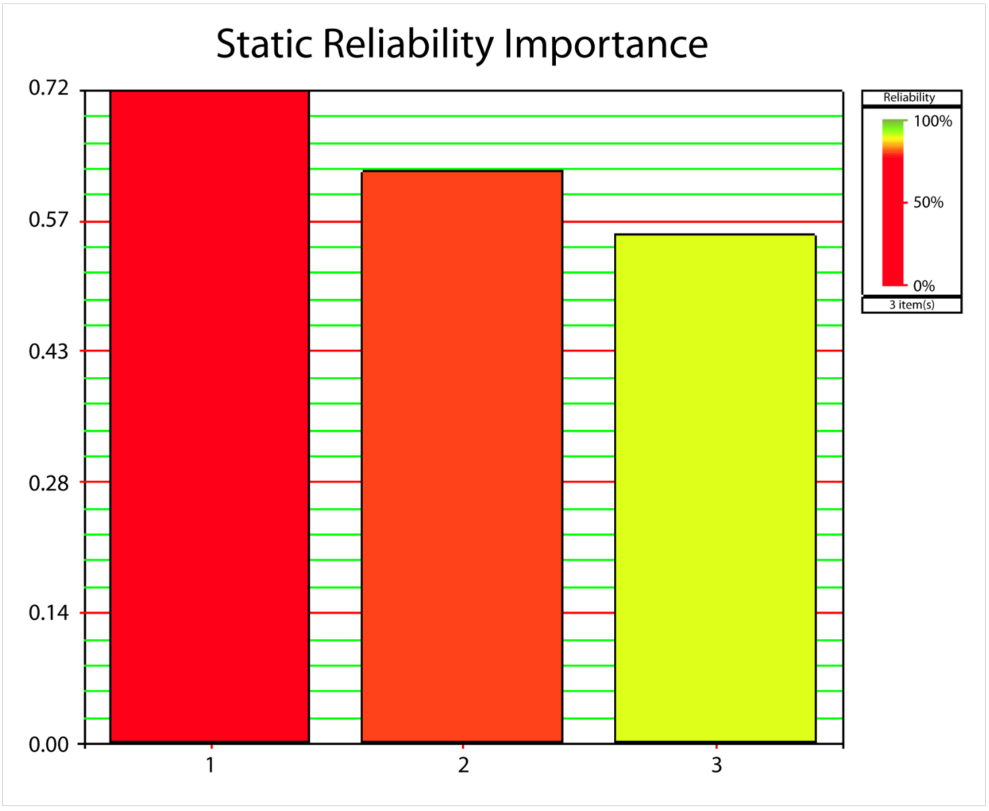
$$\begin{array}{cccccc} & Case1 & Case2 & Case3 & Case4 & Case5 \\ Component1 & 0.9655 & 0.9874 & 0.9552 & 0.9790 & 0.9295 \\ Component2 & 0.9655 & 0.9633 & 0.9649 & 0.9553 & 0.9884 \\ Component3 & 0.9655 & 0.9463 & 0.9765 & 0.9624 & 0.9797 \end{array}$$

Case 2 - It can be seen that the highest reliability was allocated to Component 1 with the Easy feasibility. The lowest reliability was assigned to Component 3 with the Hard feasibility. This makes sense in that an optimized reliability scheme will call for the greatest reliability changes in those components that are the easiest to change.

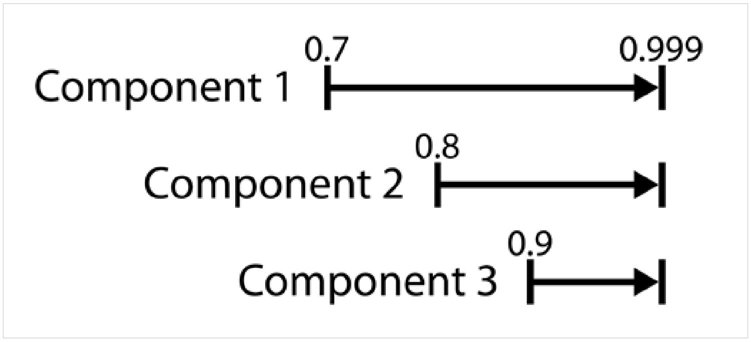
Case 3 - The components were different but had the same feasibility values.

In other words, all three components have the same opportunity for improvement. This case differs from Cases 1 and 2 since there are two factors, not present previously, that will affect the outcome of the allocation in this case. First, each component in this case has a different reliability importance (impact of a component on the system's reliability); whereas in Cases 1 and 2, all three components were identical and had the same reliability importance.

The next figure shows the reliability importance for each component, where it can be seen that Component 1 has the greatest reliability importance and Component 3 has the smallest (this reliability importance also applies in Cases 4 and 5). This indicates that the reliability of Component 1 should be significantly increased because it has the biggest impact on the overall system reliability.



In addition, each component's cost function in Case 3 also depends on the difference between each component's initial reliability and its corresponding maximum achievable reliability. (In Cases 1 and 2 this was not an issue because the components were identical.) The greater this difference, the greater the cost of improving the reliability of a particular component relative to the other two components. This difference between the initial reliability of a component and its maximum achievable reliability is called the range of improvement for that component. Since all three components have the same maximum achievable reliability, Component 1, with the largest range for improvement, is the most cost efficient component to improve. The improvement ranges for all three components are illustrated in the next figure. At the same time, however, there is a reliability value between the initial and the maximum achievable reliability beyond which it becomes cost prohibitive to improve any further. This reliability value is dictated by the feasibility value. From the table of results, it can be seen that in Case 3 there was a 25.52% improvement for Component 1, 16.49% for Component 2 and 7.65% for Component 3.



Case 4 - As opposed to Case 3, Component 1 was assigned an even greater increase of 27.9%, with Components 2 and 3 receiving lesser increases than in Case 3, of 15.53% and 6.24% respectively. This is due to the fact that Component 1 has an Easy feasibility and Component 3 has a Hard feasibility, which means that it is more difficult to increase the reliability of Component 3 than to increase the reliability of Component 1.

Case 5 - The feasibility values here are reversed with Component 1 having a Hard feasibility and Component 3 an Easy feasibility. The recommended increase in Component 1's reliability is less compared to its increase for Cases 3 and 4. Note, however, that Components 2 and 3 still received a smaller increase in reliability than Component 1 because their ranges of improvement are smaller. In other words, Component 3 was assigned the smallest increase in reliability in Cases 3, 4 and 5 because its initial reliability is very close to its maximum achievable reliability.

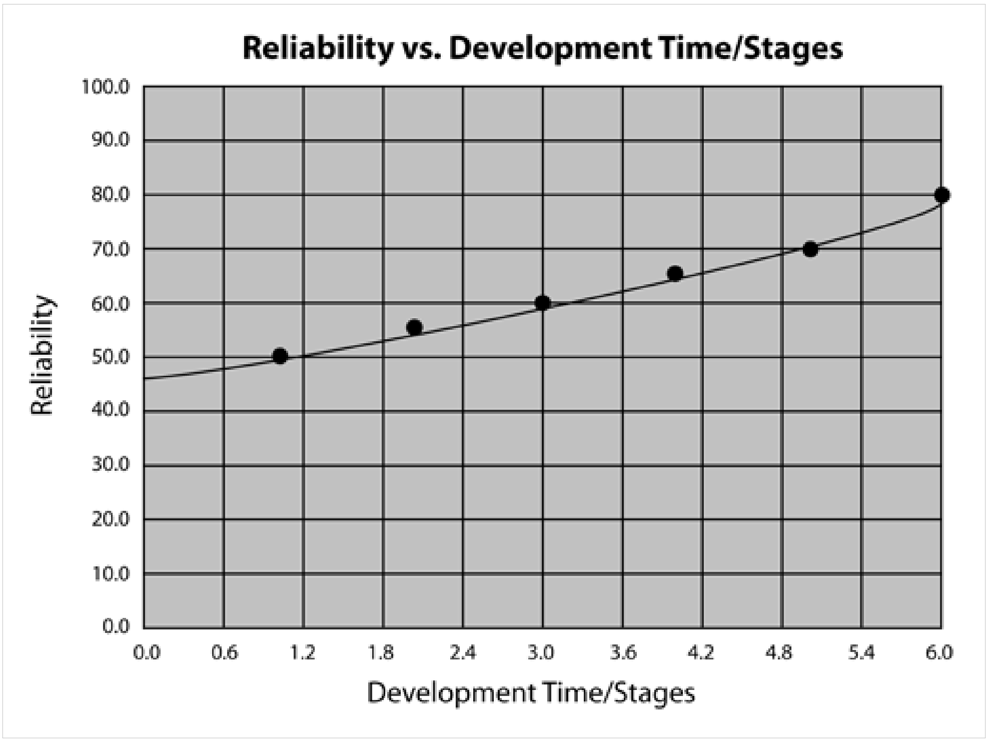
Setting Specifications

This methodology could also be used to arrive at initial specifications for a set of components. In the prior examples, we assumed a current reliability for the components. One could repeat these steps by choosing an arbitrary (lower) initial reliability for each component, thus allowing the algorithm to travel up to the target. When doing this, it is important to keep in mind the fact that both the distance from the target (the distance from the initial arbitrary value and the target value) for each component is also a significant contributor to the final results, as presented in the prior example. If one wishes to arrive at the results using only the cost functions then it may be advantageous to set equal initial reliabilities for all components.

Other Notes on User-Defined Cost Functions

The optimization method in BlockSim is a very powerful tool for allocating reliability to the components of a system while minimizing an overall cost of improvement. The default cost function in BlockSim was derived in order to model a general relationship between the cost and the component reliability. However, if actual cost information is available, then one can use the cost data instead of using the default function. Additionally, one can also view the feasibility in the default function as a measure of the difficulty in increasing the reliability of the component relative to the rest of the components to be optimized, assuming that they also follow the same cost function with the corresponding feasibility values. If fault tolerance is a viable option, a reliability cost function for adding parallel units can be developed as demonstrated previously.

Another method for developing a reliability cost function would be to obtain different samples of components from different suppliers and test the samples to determine the reliability of each sample type. From this data, a curve could be fitted through standard regression techniques and an equation defining the cost as a function of reliability could be developed. The following figure shows such a curve.



Lastly, and in cases where a reliability growth program is in place, the simplest way of obtaining a relationship between cost and reliability is by associating a cost to each development stage of the growth process. Reliability growth models such as the Crow (AMSAA), Duane, Gompertz and Logistic models can be used to describe the cost as a function of reliability.

If a reliability growth model has been successfully implemented, the development costs over the respective development time stages can be applied to the growth model, resulting in equations that describe reliability/cost relationships. These equations can then be entered into BlockSim as user-defined cost functions (feasibility policies). The only potential drawback to using growth model data is the lack of flexibility in applying the optimum results. Making the cost projection for future stages of the project would require the assumption that development costs will be accrued at a similar rate in the future, which may not always be a valid assumption. Also, if the optimization result suggests using a high reliability value for a component, it may take more time than is allotted for that project to attain the required reliability given the current reliability growth of the project.

References

[1] http://www.reliasoft.com/xfmea/index.htm

Chapter 6

Introduction to Repairable Systems

In prior chapters, the analysis was focused on determining the reliability of the system (i.e., the probability that the system, subsystem or component will operate successfully by a given time, t. The prior formulations provided us with the probability of success of the entire system, up to a point in time, without looking at the question: "What happens if a component fails during that time and is then fixed?" In dealing with repairable systems, these definitions need to be redefined and adapted to deal with this case of the renewal of systems/components.

Repairable systems receive maintenance actions that restore/renew system components when they fail. These actions change the overall makeup of the system. These actions must now be taken into consideration when assessing the behavior of the system because the age of the system components is no longer uniform nor is the time of operation of the system continuous.

In attempting to understand the system behavior, additional information and models are now needed for each system component. Our primary input in the prior chapters was a model that described how the component failed (its failure probability distribution). When dealing with components that are repaired, one also needs to know how long it takes for the component to be restored. That is, at the very least, one needs a model that describes how the component is restored (a repair probability distribution).

In this chapter, we will introduce the additional information, models and metrics required to fully analyze a repairable system.

Defining Maintenance

To properly deal with repairable systems, we need to first understand how components in these systems are restored (i.e., the maintenance actions that the components undergo). In general, maintenance is defined as any action that restores failed units to an operational condition or retains non-failed units in an operational state. For repairable systems, maintenance plays a vital role in the life of a system. It affects the system's overall reliability, availability, downtime, cost of operation, etc. Generally, maintenance actions can be divided into three types: corrective maintenance, preventive maintenance and inspections.

Corrective Maintenance

Corrective maintenance consists of the action(s) taken to restore a failed system to operational status. This usually involves replacing or repairing the component that is responsible for the failure of the overall system. Corrective maintenance is performed at unpredictable intervals because a component's failure time is not known *a priori*. The objective of corrective maintenance is to restore the system to satisfactory operation within the shortest possible time. Corrective maintenance is typically carried out in three steps:

- Diagnosis of the problem. The maintenance technician must take time to locate the failed parts or otherwise satisfactorily assess the cause of the system failure.
- Repair and/or replacement of faulty component(s). Once the cause of system failure has been determined, action must be taken to address the cause, usually by replacing or repairing the components that caused the system to fail.
- Verification of the repair action. Once the components in question have been repaired or replaced, the maintenance technician must verify that the system is again successfully operating.

Preventive Maintenance

Preventive maintenance, unlike corrective maintenance, is the practice of replacing components or subsystems before they fail in order to promote continuous system operation. The schedule for preventive maintenance is based on observation of past system behavior, component wear-out mechanisms and knowledge of which components are vital to continued system operation. Cost is always a factor in the scheduling of preventive maintenance. In many circumstances, it is financially more sensible to replace parts or components that have not failed at predetermined intervals rather than to wait for a system failure that may result in a costly disruption in operations. Preventive maintenance scheduling strategies are discussed in more detail later in this chapter.

Inspections

Inspections are used in order to uncover hidden failures (also called dormant failures). In general, no maintenance action is performed on the component during an inspection unless the component is found failed, in which case a corrective maintenance action is initiated. However, there might be cases where a partial restoration of the inspected item would be performed during an inspection. For example, when checking the motor oil in a car between scheduled oil changes, one might occasionally add some oil in order to keep it at a constant level. The subject of inspections is discussed in more detail in Repairable Systems Analysis Through Simulation.

Downtime Distributions

Maintenance actions (preventive or corrective) are not instantaneous. There is a time associated with each action (i.e., the amount of time it takes to complete the action). This time is usually referred to as downtime and it is defined as the length of time an item is not operational. There are a number of different factors that can affect the length of downtime, such as the physical characteristics of the system, spare part availability, repair crew availability, human factors, environmental factors, etc. Downtime can be divided into two categories based on these factors:

- **Waiting Downtime.** This is the time during which the equipment is inoperable but not yet undergoing repair. This could be due to the time it takes for replacement parts to be shipped, administrative processing time, etc.
- **Active Downtime.** This is the time during which the equipment is inoperable and actually undergoing repair. In other words, the active downtime is the time it takes repair personnel to perform a repair or replacement. The length of the active downtime is greatly dependent on human factors, as well as on the design of the equipment. For example, the ease of accessibility of components in a system has a direct effect on the active downtime.

These downtime definitions are subjective and not necessarily mutually exclusive nor all-inclusive. As an example, consider the time required to diagnose the problem. One may need to diagnose the problem before ordering parts and then wait for the parts to arrive.

The influence of a variety of different factors on downtime results in the fact that the time it takes to repair/restore a specific item is not generally constant. That is, the time-to-repair is a random variable, much like the time-to-failure. The statement that it takes on average five hours to repair implies an underlying probabilistic distribution. Distributions that describe the time-to-repair are called repair distributions (or downtime distributions) in order to distinguish them from the failure distributions. However, the methods employed to quantify these distributions are not any different mathematically than the methods employed to quantify failure distributions. The difference is in how they are employed (i.e., the events they describe and metrics used). As an example, when using a life distribution with failure data (i.e., the event modeled was time-to-failure), unreliability provides the probability that the event (failure) will occur by that time, while reliability provides the probability that the event (failure) will not occur. In the case of downtime distributions, the data set consists of times-to-repair, thus what we termed as unreliability now becomes the probability of the event occurring (i.e., repairing the component). Using these definitions, the probability of repairing the component by a given time, t, is also called the component's maintainability.

Maintainability

Maintainability is defined as the probability of performing a successful repair action within a given time. In other words, maintainability measures the ease and speed with which a system can be restored to operational status after a failure occurs. For example, if it is said that a particular component has a 90% maintainability in one hour, this means that there is a 90% probability that the component will be repaired within an hour. In maintainability, the random variable is time-to-repair, in the same manner as time-to-failure is the random variable in reliability. As an example, consider the maintainability equation for a system in which the repair times are distributed exponentially. Its maintainability $M\left(t\right)$is given by:

$$M\left(t\right)=1-e^{-\mu \cdot t}$$

where μ= repair rate.

Note the similarity between this equation and the equation for the reliability of a system with exponentially distributed failure times. However, since the maintainability represents the probability of an event occurring (repairing the system) while the reliability represents the probability of an event not occurring (failure), the maintainability expression is the equivalent of the unreliability expression, $(1-R)$. Furthermore, the single model parameter $\mu$is now referred to as the repair rate, which is analogous to the failure rate, λ, used in reliability for an exponential distribution.

Similarly, the mean of the distribution can be obtained by:

$$\frac{1}{\mu}=MTTR(\text{mean time to repair})$$

This now becomes the mean time to repair ($MTTR$) instead of the mean time to failure ($MTTF$).

The same concept can be expanded to other distributions. In the case of the Weibull distribution, maintainability, $M\left(t\right)$, is given by:

$$M(t)=1-e^{-\left(\frac{t}{\eta}\right)^{\beta}}$$

While the mean time to repair ($MTTR$) is given by:

$$MTTR\quad =\eta \cdot \Gamma \left(\frac{1}{\beta}+1\right)$$

And the Weibull repair rate is given by:

$$\mu(t)=\frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}$$

As a last example, if a lognormal distribution is chosen, then:

$$M(t)=\int_{0}^{T'}\frac{1}{\sigma_{T'}\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{t-\overline{T'}}{\sigma_{T'}}\right)^{2}}dt$$

where:

- $\bar{T}'$ =mean of the natural logarithms of the times-to-repair.
- $\sigma_{T'}$ =standard deviation of the natural logarithms of the times-to-repair.

It should be clear by now that any distribution can be used, as well as related concepts and methods used in life data analysis. The only difference being that instead of times-to-failure we are using times-to-repair. What one chooses to include in the time-to-repair varies, but can include:

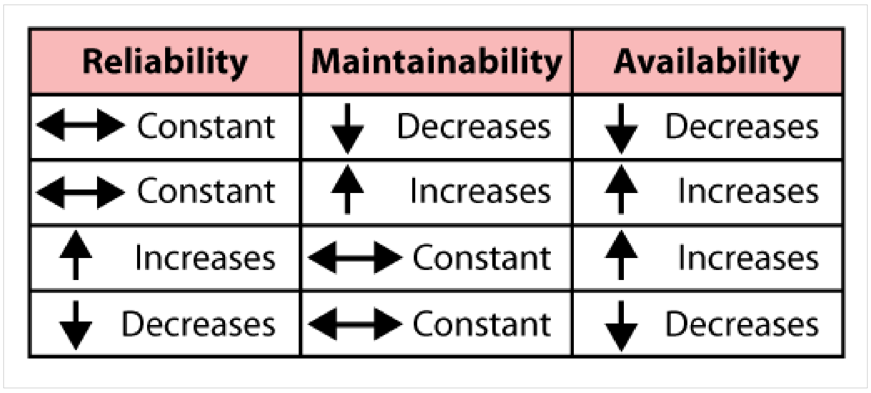
1. The time it takes to successfully diagnose the cause of the failure.
2. The time it takes to procure or deliver the parts necessary to perform the repair.

3. The time it takes to gain access to the failed part or parts.
4. The time it takes to remove the failed components and replace them with functioning ones.
5. The time involved with bringing the system back to operating status.
6. The time it takes to verify that the system is functioning within specifications.
7. The time associated with closing up a system and returning it to normal operation.

In the interest of being fair and accurate, one should disclose (document) what was and was not included in determining the repair distribution.

Availability

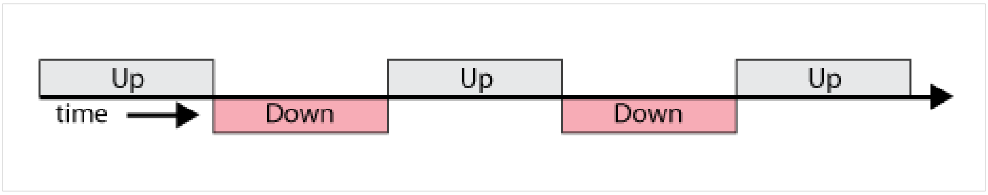
If one considers both reliability (probability that the item will not fail) and maintainability (the probability that the item is successfully restored after failure), then an additional metric is needed for the probability that the component/system is operational at a given time, t(i.e., has not failed or it has been restored after failure). This metric is availability. Availability is a performance criterion for repairable systems that accounts for both the reliability and maintainability properties of a component or system. It is defined as the probability that the system is operating properly when it is requested for use. That is, availability is the probability that a system is not failed or undergoing a repair action when it needs to be used. For example, if a lamp has a 99.9% availability, there will be one time out of a thousand that someone needs to use the lamp and finds out that the lamp is not operational either because the lamp is burned out or the lamp is in the process of being replaced. Note that this metric alone tells us nothing about how many times the lamp has been replaced. For all we know, the lamp may be replaced every day or it could have never been replaced at all. Other metrics are still important and needed, such as the lamp's reliability. The next table illustrates the relationship between reliability, maintainability and availability.



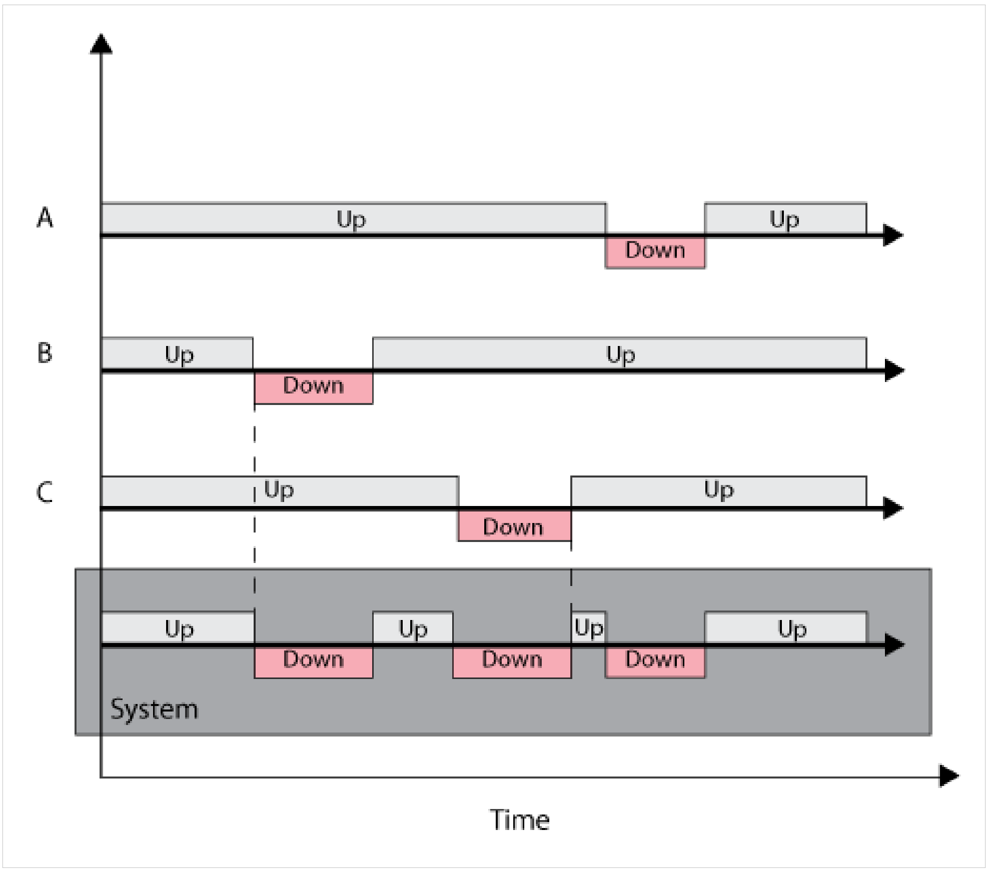
Reliability	Maintainability	Availability
↔ Constant	↓ Decreases	↓ Decreases
↔ Constant	↑ Increases	↑ Increases
↑ Increases	↔ Constant	↑ Increases
↓ Decreases	↔ Constant	↓ Decreases

A Brief Introduction to Renewal Theory

For a repairable system, the time of operation is not continuous. In other words, its life cycle can be described by a sequence of up and down states. The system operates until it fails, then it is repaired and returned to its original operating state. It will fail again after some random time of operation, get repaired again, and this process of failure and repair will repeat. This is called a renewal process and is defined as a sequence of independent and non-negative random variables. In this case, the random variables are the times-to-failure and the times-to-repair/restore. Each time a unit fails and is restored to working order, a renewal is said to have occurred. This type of renewal process is known as an alternating renewal process because the state of the component alternates between a functioning state and a repair state, as illustrated in the following graphic.



A system's renewal process is determined by the renewal processes of its components. For example, consider a series system of three statistically independent components. Each component has a failure distribution and a repair distribution. Since the components are in series, when one component fails, the entire system fails. The system is then down for as long as the failed component is under repair. The following figure illustrates this.



One of the main assumptions in renewal theory is that the failed components are replaced with new ones or are repaired so they are as good as new, hence the name renewal. One can make the argument that this is the case for every repair, if you define the system in enough detail. In other words, if the repair of a single circuit board in the system involves the replacement of a single transistor in the offending circuit board, then if the analysis (or RBD) is performed down to the transistor level, the transistor itself gets renewed. In cases where the analysis is done at a higher level, or if the offending component is replaced with a used component, additional steps are required. We will discuss this in later chapters using a restoration factor in the analysis. For more details on renewal theory, interested readers can refer to Elsayed [7] and Leemis [17].

Availability Classifications

The definition of availability is somewhat flexible and is largely based on what types of downtimes one chooses to consider in the analysis. As a result, there are a number of different classifications of availability, such as:

Instantaneous or Point Availability, $A\left(t\right)$

Instantaneous (or point) availability is the probability that a system (or component) will be operational (up and running) at any random time, t. This is very similar to the reliability function in that it gives a probability that a system will function at the given time, t. Unlike reliability, the instantaneous availability measure incorporates maintainability information. At any given time, t, the system will be operational if the following conditions are met Elsayed ([7]):

The item functioned properly from $0$to $t$with probability $R(t)$or it functioned properly since the last repair at time u, $0 < u < t$, with probability:

$$\int_0^t R(t-u)m(u)du$$

With $m(u)$being the renewal density function of the system.

Then the point availability is the summation of these two probabilities, or:

$$A\left(t\right) = R(t) + \int_0^t R(t-u)m(u)du$$

Average Uptime Availability (or Mean Availability), $\overline{A}\left(t\right)$

The mean availability is the proportion of time during a mission or time period that the system is available for use. It represents the mean value of the instantaneous availability function over the period (0, T] and is given by:

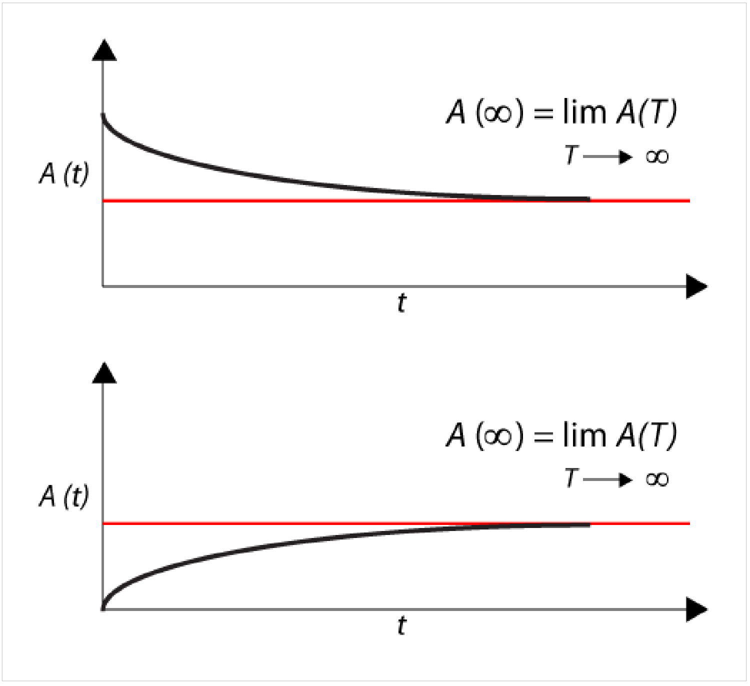
$$\overline{A\left(t\right)} = \frac{1}{t}\int_0^t A\left(u\right)du$$

Steady State Availability, $A(\infty)$

The steady state availability of the system is the limit of the instantaneous availability function as time approaches infinity or:

$$A(\infty) = \lim_{t\to\infty} A(t)$$

The figure shown next also graphically illustrates this.



In other words, one can think of the steady state availability as a stabilizing point where the system's availability is a constant value. However, one has to be very careful in using the steady state availability as the sole metric for some systems, especially systems that do not need regular maintenance. A large scale system with repeated repairs, such as a car, will reach a point where it is almost certain that something will break and need repair once a month. However, this state may not be reached until, say, 500,000 miles. Obviously, if I am an operator of rental vehicles and I only keep the vehicles until they reach 50,000 miles, then this value would not be of any use to me. Similarly, if I am an auto maker and only warrant the vehicles to Xmiles, is knowing the steady state value useful?

Inherent Availability, A_I

Inherent availability is the steady state availability when considering only the corrective downtime of the system.

For a single component, this can be computed by:

$$A_I = \frac{MTTF}{MTTF + MTTR}$$

This gets slightly more complicated for a system. To do this, one needs to look at the mean time between failures, or $MTBF$, and compute this as follows:

$$A_I = \frac{MTBF}{MTBF + MTTR}$$

This may look simple. However, one should keep in mind that until the steady state is reached, the $MTBF$may be a function of time (e.g., a degrading system), thus the above formulation should be used cautiously. Furthermore, it is important to note that the $MTBF$defined here is different from the $MTTF$(or more precisely for a repairable system, $MTTFF$, mean time to first failure).

Achieved Availability, A_A

Achieved availability is very similar to inherent availability with the exception that preventive maintenance (PM) downtimes are also included. Specifically, it is the steady state availability when considering corrective and preventive downtime of the system. It can be computed by looking at the mean time between maintenance actions, $MTBM$,and the mean maintenance downtime, $\overline{M}$,or:

$$A_A = \frac{MTBM}{MTBM + \overline{M}}$$

Operational Availability, A_o

Operational availability is a measure of the average availability over a period of time and it includes all experienced sources of downtime, such as administrative downtime, logistic downtime, etc.

Operational availability is the ratio of the system uptime and total time. Mathematically, it is given by:

$$A_o = \frac{Uptime}{Operating\ Cycle}$$

Where the operating cycle is the overall time period of operation being investigated and uptime is the total time the system was functioning during the operating cycle.

When there is no specified logistic downtime or preventive maintenance, the above equation returns the Mean Availability of the system. The operational availability is the availability that the customer actually experiences. It is essentially the *a posteriori* availability based on actual events that happened to the system. The previous availability definitions are *a priori* estimations based on models of the system failure and downtime distributions. In many cases, operational availability cannot be controlled by the manufacturer due to variation in location, resources and other factors that are the sole province of the end user of the product.

Example: Calculating Availability

As an example, consider the following scenario. A diesel power generator is supplying electricity at a research site in Antarctica. The personnel are not satisfied with the generator. They estimated that in the past six months, they were without electricity due to generator failure for an accumulated time of 1.5 months. Therefore, the operational availability of the diesel generator experienced by the personnel of the station is:

$$A_o = \frac{4.5\text{ months}}{6\text{ months}} = 0.75$$

Obviously, this is not satisfactory performance for an electrical generator in such a climate, so alternatives to this source of electricity are investigated. One alternative under consideration is a wind-powered electrical turbine, which the manufacturer claims to have a 99.71% availability. This is much higher than the availability experienced by the crew of the Antarctic research station for the diesel generator. Upon investigation, it was found that the wind-turbine manufacturer estimated the availability based on the following information:

Failure Distribution	Repair Distribution
Exponential, $MTTF = 2400\,hr$	Exponential, $MTTR = 7\,hr$

Based on the above information, one can estimate the mean availability for the wind turbine over a period of six months to be:

$$\overline{A} = A_I = 0.9972$$

This availability, however, was obtained solely by considering the claimed failure and repair properties of the wind-turbine. Waiting downtime was not considered in the above calculation. Therefore, this availability measure cannot be compared to the operational availability for the diesel generator since the two availability measurements have different inputs. This form of availability measure is also known as inherent availability. In order to make a meaningful comparison, the inherent availability of the diesel generator needs to be estimated. The diesel generator has an $MTTF$= 50 days (or 1200 hours) and an $MTTR$= 3 hours. Thus, an estimate of the mean availability is:

$$\overline{A} = A_I = 0.9975$$

Note that the inherent availability of the diesel generator is actually a little bit better than the inherent availability of the wind-turbine! Even though the diesel generator has a higher failure rate, its mean-time-to-repair is much smaller than that of the wind turbine, resulting in a slightly higher inherent availability value. This example illustrates the potentially large differences in the types of availability measurements, as well as their misuse. In this example, the operational availability is much lower than the inherent availability. This is because the inherent availability does not account for downtime due to administrative time, logistic time, the time required to obtain spare parts or the time it takes for the repair personnel to arrive at the site.

Preventive Maintenance

Preventive maintenance (PM) is a schedule of planned maintenance actions aimed at the prevention of breakdowns and failures. The primary goal of preventive maintenance is to prevent the failure of equipment before it actually occurs. It is designed to preserve and enhance equipment reliability by replacing worn components before they actually fail. Preventive maintenance activities include equipment checks, partial or complete overhauls at specified periods, oil changes, lubrication and so on. In addition, workers can record equipment deterioration so they know to replace or repair worn parts before they cause system failure. Recent technological advances in tools for inspection and diagnosis have enabled even more accurate and effective equipment maintenance. The ideal preventive maintenance program would prevent all equipment failure before it occurs.

Value of Preventive Maintenance

There are multiple misconceptions about preventive maintenance. One such misconception is that PM is unduly costly. This logic dictates that it would cost more for regularly scheduled downtime and maintenance than it would normally cost to operate equipment until repair is absolutely necessary. This may be true for some components; however, one should compare not only the costs but the long-term benefits and savings associated with preventive maintenance. Without preventive maintenance, for example, costs for lost production time from unscheduled equipment breakdown will be incurred. Also, preventive maintenance will result in savings due to an increase of effective system service life.

Long-term benefits of preventive maintenance include:

- Improved system reliability.
- Decreased cost of replacement.
- Decreased system downtime.
- Better spares inventory management.

Long-term effects and cost comparisons usually favor preventive maintenance over performing maintenance actions only when the system fails.

When Does Preventive Maintenance Make Sense?

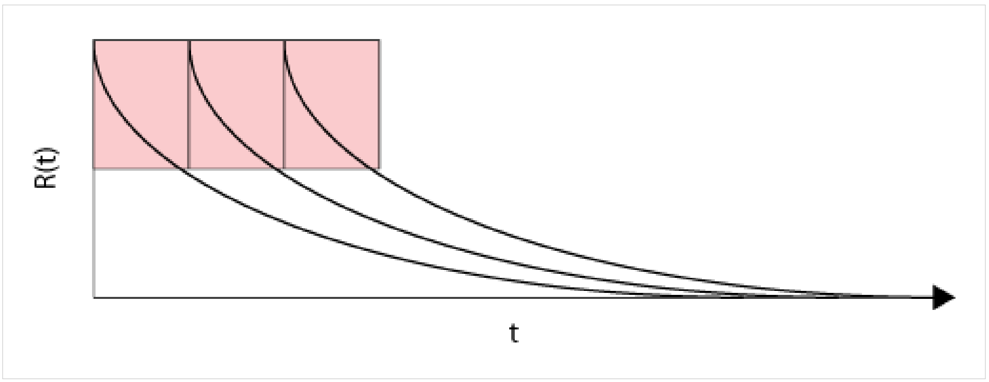
Preventive maintenance is a logical choice if, and only if, the following two conditions are met:

- Condition #1: The component in question has an increasing failure rate. In other words, the failure rate of the component increases with time, implying wear-out. Preventive maintenance of a component that is assumed to have an exponential distribution (which implies a constant failure rate) does not make sense!
- Condition #2: The overall cost of the preventive maintenance action must be less than the overall cost of a corrective action.

If both of these conditions are met, then preventive maintenance makes sense. Additionally, based on the costs ratios, an optimum time for such action can be easily computed for a single component. This is detailed in later sections.

The Fallacy of "Constant Failure Rate" and "Preventive Replacement"

Even though we alluded to the fact in the last section, it is important to make it explicitly clear that if a component has a constant failure rate (i.e., defined by an exponential distribution), then preventive maintenance of the component will have no effect on the component's failure occurrences. To illustrate this, consider a component with an $MTTF = 100$ hours, or $\lambda = 0.01$, and with preventive replacement every 50 hours. The reliability vs. time graph for this case is illustrated in the following figure, where the component is replaced every 50 hours, thereby resetting the component's reliability to one. At first glance, it may seem that the preventive maintenance action is actually maintaining the component at a higher reliability.



However, consider the following cases for a single component:

Case 1: The component's reliability from 0 to 60 hours:

- With preventive maintenance, the component was replaced with a new one at 50 hours so the overall reliability is based on the reliability of the new component for 10 hours, $R(t=10)=90.48\%$, times the reliability of the previous component, $R(t=50)=60.65\%$. The result is $R(t=60)=54.88\%$.
- Without preventive maintenance, the reliability would be the reliability of the same component operating to 60 hours, or $R(t=60)=54.88\%$.

Case 2: The component's reliability from 50 to 60 hours:

- With preventive maintenance, the component was replaced at 50 hours, so this is solely based on the reliability of the new component for a mission of 10 hours, or $R(t=10)=90.48\%$.
- Without preventive maintenance, the reliability would be the conditional reliability of the same component operating to 60 hours, having already survived to 50 hours, or $R_C(T=60|50)=R(60)/R(50)=90.48\%$.

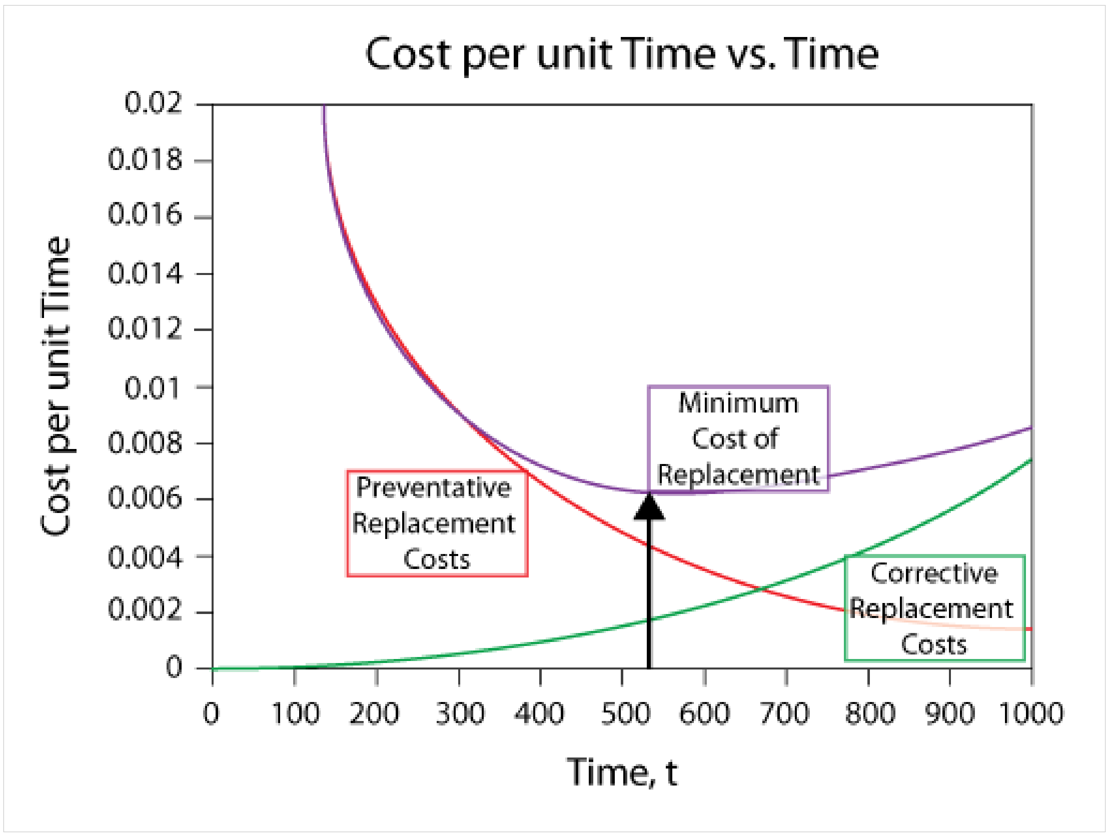
As can be seen, both cases — with and without preventive maintenance — yield the same results.

Determining Preventive Replacement Time

As mentioned earlier, if the component has an increasing failure rate, then a carefully designed preventive maintenance program is beneficial to system availability. Otherwise, the costs of preventive maintenance might actually outweigh the benefits. The objective of a good preventive maintenance program is to either minimize the overall costs (or downtime, etc.) or meet a reliability objective. In order to achieve this, an appropriate interval (time) for scheduled maintenance must be determined. One way to do that is to use the optimum age replacement model, as presented next. The model adheres to the conditions discussed previously:

- The component is exhibiting behavior associated with a wear-out mode. That is, the failure rate of the component is increasing with time.
- The cost for planned replacements is significantly less than the cost for unplanned replacements.

The following figure shows the Cost Per Unit Time vs. Time plot and it can be seen that the corrective replacement costs increase as the replacement interval increases. In other words, the less often you perform a PM action, the higher your corrective costs will be. Obviously, as we let a component operate for longer times, its failure rate increases to a point that it is more likely to fail, thus requiring more corrective actions. The opposite is true for the preventive replacement costs. The longer you wait to perform a PM, the less the costs; if you do PM too often, the costs increase. If we combine both costs, we can see that there is an optimum point that minimizes the costs. In other words, one must strike a balance between the risk (costs) associated with a failure while maximizing the time between PM actions.



Optimum Age Replacement Policy

To determine the optimum time for such a preventive maintenance action (replacement), we need to mathematically formulate a model that describes the associated costs and risks. In developing the model, it is assumed that if the unit fails before time t, a corrective action will occur and if it does not fail by time t, a preventive action will occur. In other words, the unit is replaced upon failure or after a time of operation, t, whichever occurs first. Thus, the optimum replacement time can be found by minimizing the cost per unit time, $CPUT\left(t\right)$. $CPUT\left(t\right)$is given by:

$$\begin{aligned} CPUT\left(t\right) = & \frac{\text{Total Expected Replacement Cost per Cycle}}{\text{Expected Cycle Length}} \\ = & \frac{C_P\cdot R\left(t\right)+C_U\cdot\left[1-R\left(t\right)\right]}{\int_0^t R\left(s\right)ds} \end{aligned}$$

where:

- $R(t)$= reliability at time t.
- C_P= cost of planned replacement.
- C_U= cost of unplanned replacement.

The optimum replacement time interval, t, is the time that minimizes $CPUT\left(t\right)$. This can be found by solving for $t$such that:

$$\frac{\partial\left[CPUT(t)\right]}{\partial t}=0$$

Or by solving for a $t$that satisfies the following equation:

$$\frac{\partial\left[\frac{C_P\cdot R(t)+C_U\cdot[1-R(t)]}{\int_0^t R(s)ds}\right]}{\partial t}=0$$

Interested readers can refer to Barlow and Hunter [2] for more details on this model.

In BlockSim (Version 8 and above), you can use the Optimum Replacement window to determine the optimum replacement time either for an individual block or for multiple blocks in a diagram simultaneously. When working with multiple blocks, the calculations can be for individual blocks or for one or more groups of blocks. For each item that is included in the optimization calculations, you will need to specify the cost for a planned replacement and the cost for an unplanned replacement. This is done by calculating the costs for replacement based on the item settings using equations or simulation and then, if desired, manually entering any additional costs for either type of replacement in the corresponding columns of the table.

The equations used to calculate the costs of planned and unplanned tasks for each item based on its associated URD are as follows:

- For the cost of planned tasks, here denoted as PM cost:

$$\begin{aligned}\text{PM Cost} = {} & (\text{PM Down Time Rate} + \text{Block Level Down Time Rate}) \cdot (\text{MTTPM} + \text{Pool Delay} + \text{Crew Delay}) \\ & + \text{Crew Labor Rate} \cdot \text{MTTPM} + \text{Cost per PM} + \text{Cost per Pool} + \text{Cost per Crew}\end{aligned}$$

  Only PM tasks based on item age or system age (fixed or dynamic intervals) are considered. If there is more than one PM task based on item age, only the first one is considered.

- For the cost of the unplanned task, here denoted as CM cost:

$$\begin{aligned}\text{CM Cost} = {} & (\text{CM Down Time Rate} + \text{Block Level Down Time Rate}) \cdot (\text{MTTR} + \text{Pool Delay} + \text{Crew Delay}) \\ & + \text{Crew Labor Rate} \cdot \text{MTTR} + \text{Cost per CM} + \text{Cost per Pool} + \text{Cost per Crew} + \text{Block Level Cost per Failure}\end{aligned}$$

When using simulation, for costs associated with planned replacements, all preventive tasks based on item age or system age (fixed or dynamic intervals) are considered. Because each item is simulated as a system (i.e., in isolation from any other item), tasks triggered in other ways are not considered.

Example: Optimum Replacement Time

The failure distribution of a component is described by a 2-parameter Weibull distribution with $\beta = 2.5$and $\eta = 1000$hours.

- The cost for a corrective replacement is \$5.
- The cost for a preventive replacement is \$1.

Estimate the optimum replacement age in order to minimize these costs.

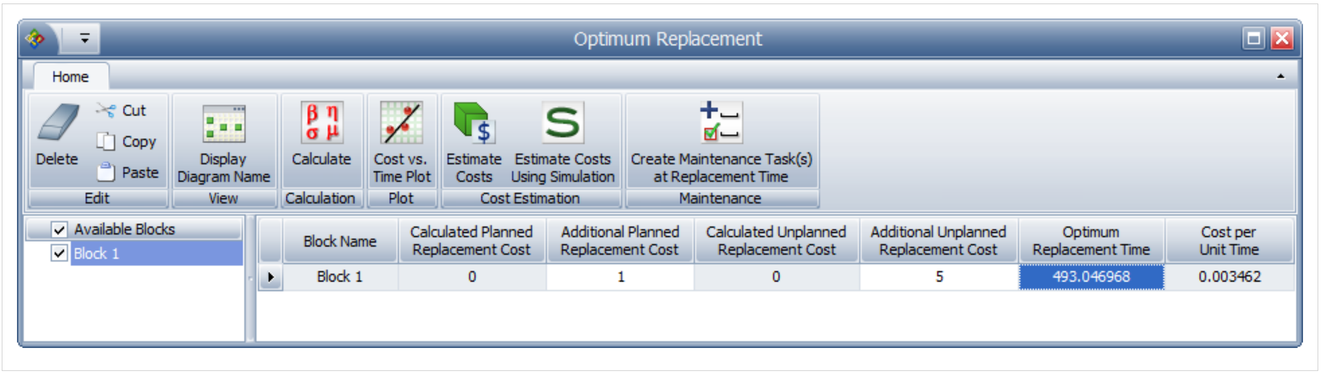
Solution

Prior to obtaining an optimum replacement interval for this component, the assumptions of the following equation must be checked.

$$\begin{aligned}CPUT(t) &= \frac{\text{Total Expected Replacement Cost per Cycle}}{\text{Expected Cycle Length}} \\ &= \frac{C_P \cdot R(t) + C_U \cdot [1 - R(t)]}{\int_0^t R(s)\,ds}\end{aligned}$$

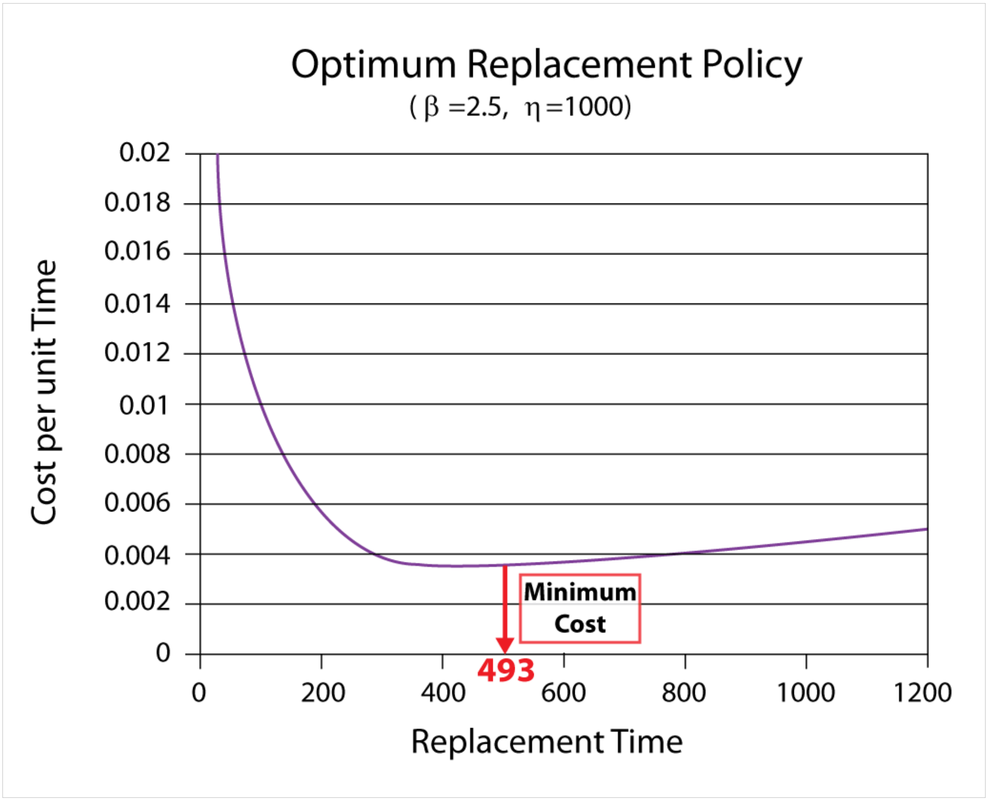
The component has an increasing failure rate because it follows a Weibull distribution with βgreater than 1. Note that if $\beta = 1$, then the component has a constant failure rate, but if $\beta < 1$, then it has a decreasing failure rate. If either of these cases exist, then preventive replacement is unwise. Furthermore, the cost for preventive replacement is less than the corrective replacement cost. Thus, the conditions for the optimum age replacement policy have been met.

Using BlockSim, enter the parameters of the Weibull distribution in the component's Block Properties window. Next, open the Optimum Replacement window and enter the **1** in the **Additional Planned Replacement Cost** column, and **5** in the **Additional Unplanned Replacement Cost** column. Click **Calculate**. In the Optimum Replacement Calculations window that appears, select the **Calculate the individual optimum replacement times** option and click **OK**. The optimum replacement time for the component is estimated to be 493.0470, as shown next.

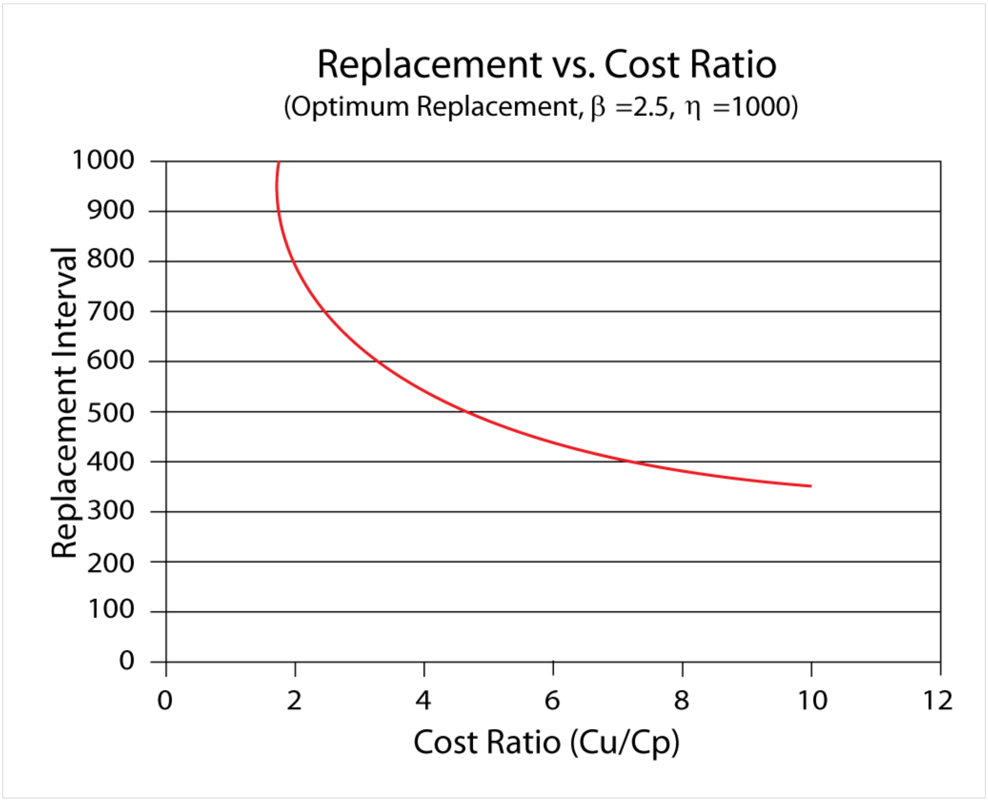


Block Name	Calculated Planned Replacement Cost	Additional Planned Replacement Cost	Calculated Unplanned Replacement Cost	Additional Unplanned Replacement Cost	Optimum Replacement Time	Cost per Unit Time
Block 1	0	1	0	5	493.046968	0.003462

The figure below shows the Cost vs. Time plot of the component (with the scaling adjusted and the plot annotated to show the minimum cost).



If we enter different cost values in the **Additional Unplanned** column and obtain the optimum replacement time at each value, we can use the data points to create a plot that shows the effect of the corrective cost on the optimum replacement interval. The following plot shows an example. In this case, the optimum replacement interval decreases as the cost ratio increases. This is an expected result because the corrective replacement costs are much greater than the preventive replacement costs. Therefore, it is more cost-effective to replace the component more frequently before it fails.



Applying These Principles to Larger Systems

In this chapter, we explored some of the basic concepts and mathematical formulations involving repairable systems. Most examples/equations were given using a single component and, in some cases, using the exponential distribution for simplicity. In practical applications where one is dealing with large systems composed of many components that fail and get repaired based on different distributions and with additional constraints (such as spare parts, crews, etc.), exact analytical computations become intractable. To solve such systems, one needs to resort to simulation (more specifically, discrete event simulation) to obtain the metrics/results discussed in this section. Repairable Systems Analysis Through Simulation expands on these concepts and introduces these simulation methods.

Chapter 7

Repairable Systems Analysis Through Simulation

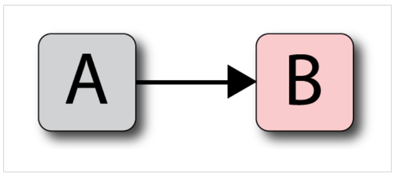
NOTE: *Some of the examples in this reference use time values with specified units (e.g., hours, days, etc.) while other examples use the abbreviation "tu" for values that could be interpreted as any given time unit. For details, see Time Units* [1].

Having introduced some of the basic theory and terminology for repairable systems in Introduction to Repairable Systems, we will now examine the steps involved in the analysis of such complex systems. We will begin by examining system behavior through a sequence of discrete deterministic events and expand the analysis using discrete event simulation.

Simple Repairs

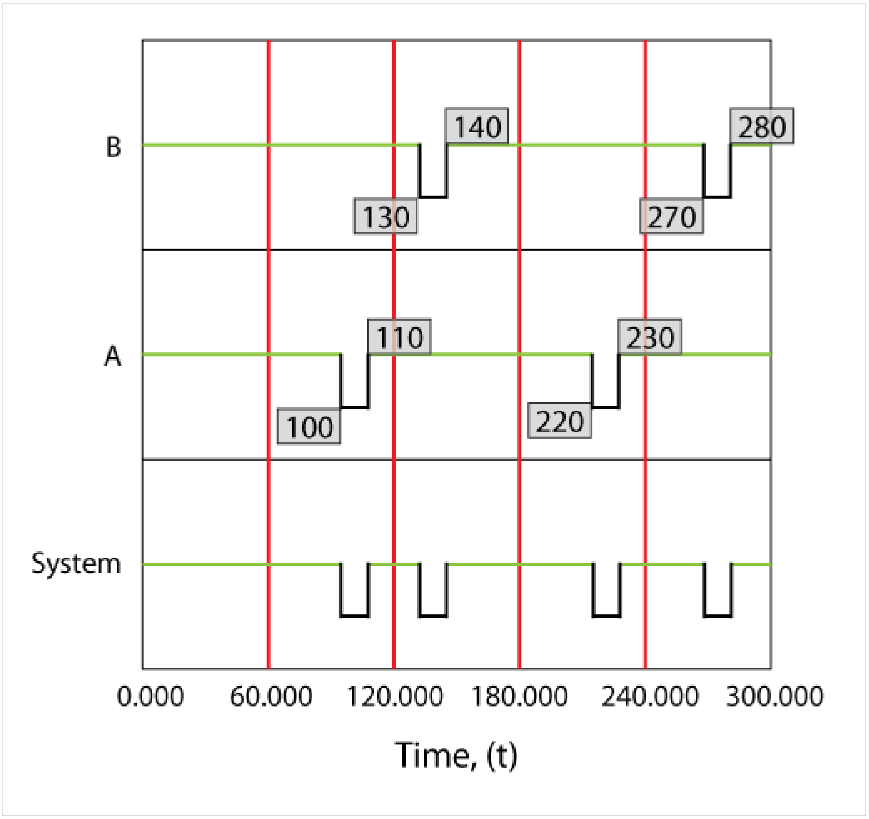
Deterministic View, Simple Series

To first understand how component failures and simple repairs affect the system and to visualize the steps involved, let's begin with a very simple deterministic example with two components, $A$and B, in series.



Component Afails every 100 hours and component Bfails every 120 hours. Both require 10 hours to get repaired. Furthermore, assume that the surviving component stops operating when the system fails (thus not aging). **NOTE**: When a failure occurs in certain systems, some or all of the system's components may or may not continue to accumulate operating time while the system is down. For example, consider a transmitter-satellite-receiver system. This is a series system and the probability of failure for this system is the probability that any of the subsystems fail. If the receiver fails, the satellite continues to operate even though the receiver is down. In this case, the continued aging of the components during the system inoperation **must** be taken into consideration, since this will affect their failure characteristics and have an impact on the overall system downtime and availability.

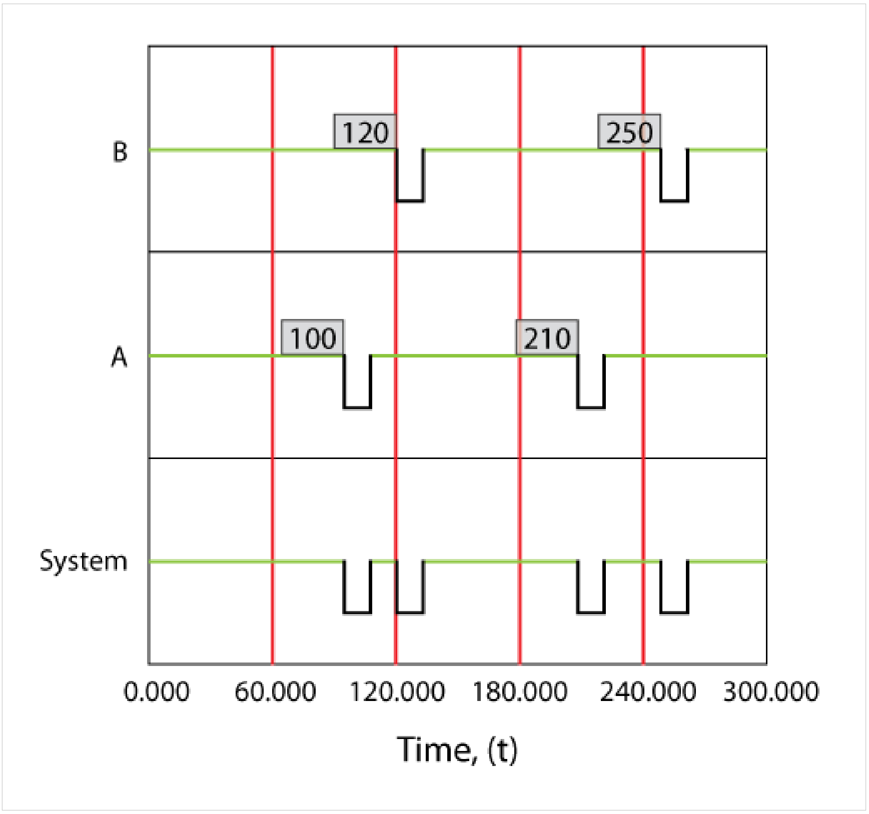
The system behavior during an operation from 0 to 300 hours would be as shown in the figure below.



Specifically, component Awould fail at 100 hours, causing the system to fail. After 10 hours, component Awould be restored and so would the system. The next event would be the failure of component B. We know that component Bfails every 120 hours (or after an age of 120 hours). Since a component does not age while the system is down, component Bwould have reached an age of 120 when the clock reaches 130 hours. Thus, component B would fail at 130 hours and be repaired by 140 and so forth. Overall in this scenario, the system would be failed for a total of 40 hours due to four downing events (two due to $A$and two due to B). The overall system availability (average or mean availability) would be $260/300 = 0.86667$. Point availability is the availability at a specific point time. In this deterministic case, the point availability would always be equal to 1 if the system is up at that time and equal to zero if the system is down at that time.

Operating Through System Failure

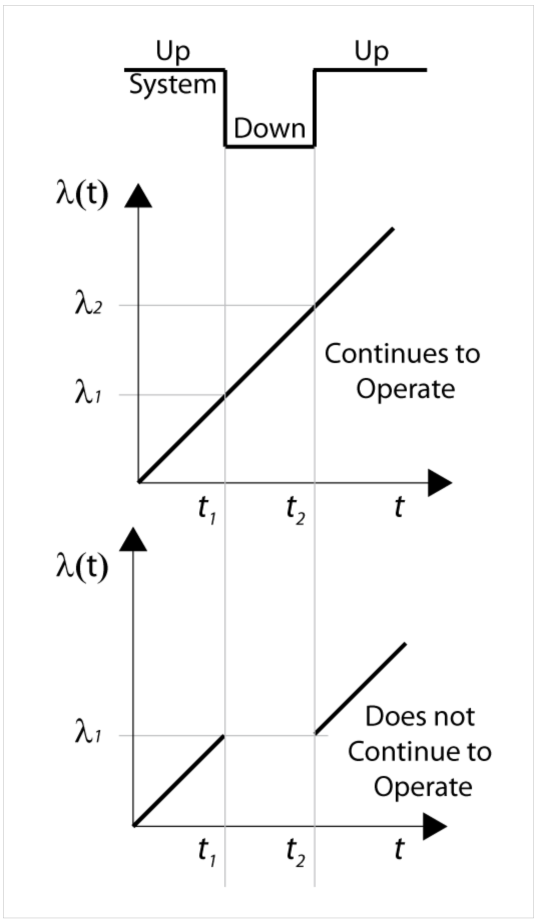
In the prior section we made the assumption that components do not age when the system is down. This assumption applies to most systems. However, under special circumstances, a unit may age even while the system is down. In such cases, the operating profile will be different from the one presented in the prior section. The figure below illustrates the case where the components operate continuously, regardless of the system status.



Effects of Operating Through Failure

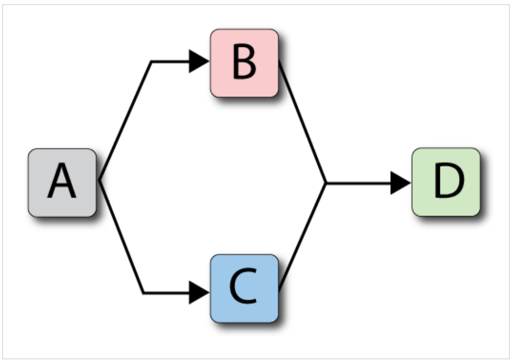
Consider a component with an increasing failure rate, as shown in the figure below. In the case that the component continues to operate through system failure, then when the system fails at $t_1$the surviving component's failure rate will be λ_1, as illustrated in figure below. When the system is restored at t_2, the component would have aged by $t_2 - t_1$and its failure rate would now be λ_2.

In the case of a component that does not operate through failure, then the surviving component would be at the same failure rate, λ_1 ,when the system resumes operation.



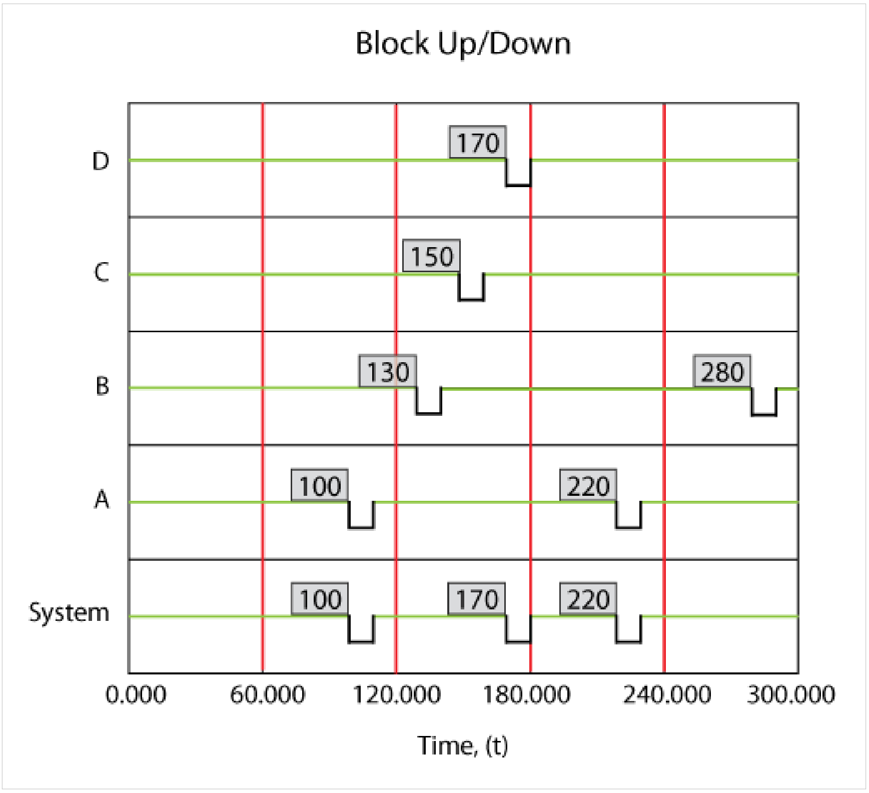
Deterministic View, Simple Parallel

Consider the following system where A fails every 100, B every 120, C every 140 and D every 160 time units. Each takes 10 time units to restore. Furthermore, assume that components do not age when the system is down.



A deterministic system view is shown in the figure below. The sequence of events is as follows:

1. At 100, A fails and is repaired by 110. The system is failed.
2. At 130, B fails and is repaired by 140. The system continues to operate.
3. At 150, C fails and is repaired by 160. The system continues to operate.
4. At 170, D fails and is repaired by 180. The system is failed.
5. At 220, A fails and is repaired by 230. The system is failed.
6. At 280, B fails and is repaired by 290. The system continues to operate.
7. End at 300.



Additional Notes

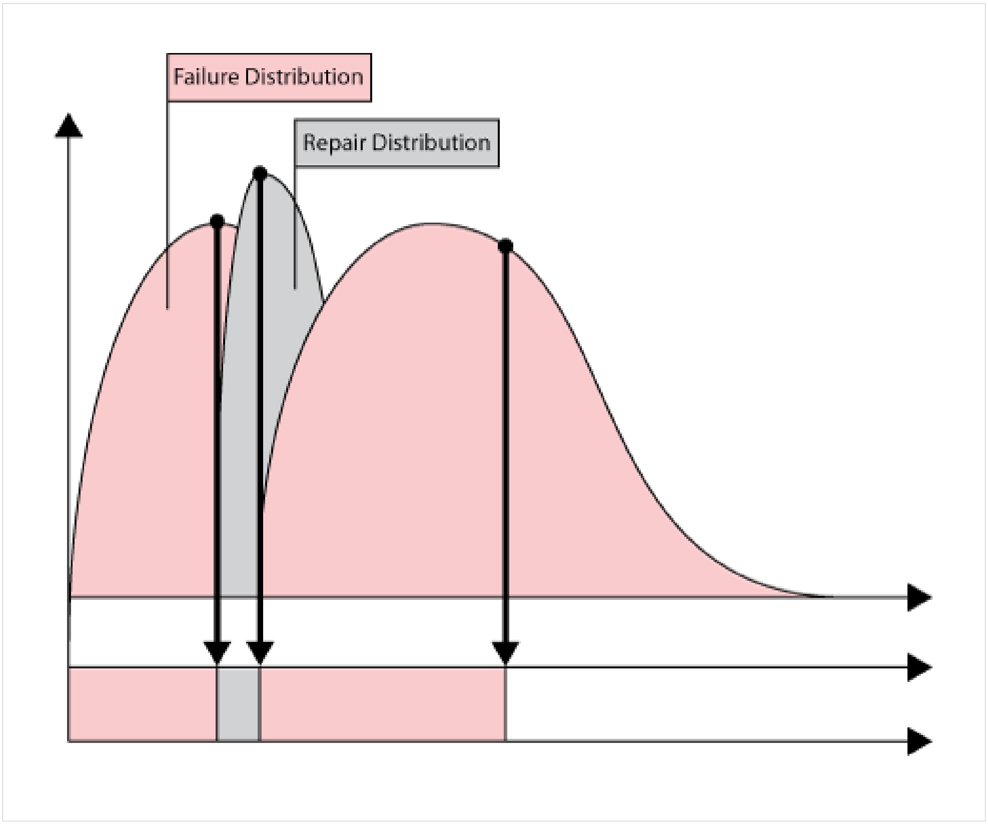
It should be noted that we are dealing with these events deterministically in order to better illustrate the methodology. When dealing with deterministic events, it is possible to create a sequence of events that one would not expect to encounter probabilistically. One such example consists of two units in series that do not operate through failure but both fail at exactly 100, which is highly unlikely in a real-world scenario. In this case, the assumption is that one of the events must occur at least an infinitesimal amount of time (dt)before the other. Probabilistically, this event is extremely rare, since both randomly generated times would have to be exactly equal to each other, to 15 decimal points. In the rare event that this happens, BlockSim would pick the unit with the lowest ID value as the first failure. BlockSim assigns a unique numerical ID when each component is created. These can be viewed by selecting the **Show Block ID** option in the Diagram Options window.

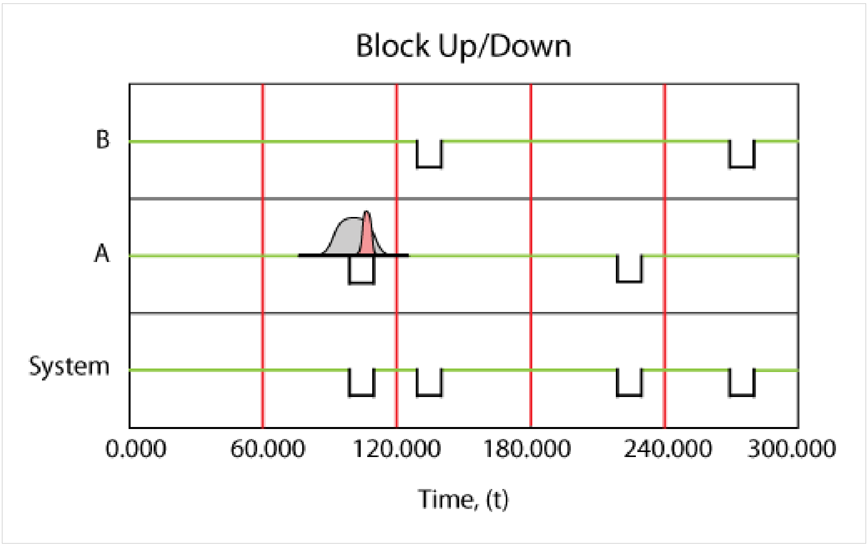
Deterministic Views of More Complex Systems

Even though the examples presented are fairly simplistic, the same approach can be repeated for larger and more complex systems. The reader can easily observe/visualize the behavior of more complex systems in BlockSim using the Up/Down plots. These are the same plots used in this chapter. It should be noted that BlockSim makes these plots available only when a single simulation run has been performed for the analysis (i.e., Number of Simulations = 1). These plots are meaningless when doing multiple simulations because each run will yield a different plot.

Probabilistic View, Simple Series

In a probabilistic case, the failures and repairs do not happen at a fixed time and for a fixed duration, but rather occur randomly and based on an underlying distribution, as shown in the following figures.





We use discrete event simulation in order to analyze (understand) the system behavior. Discrete event simulation looks at each system/component event very similarly to the way we looked at these events in the deterministic example. However, instead of using deterministic (fixed) times for each event occurrence or duration, random times are used. These random times are obtained from the underlying distribution for each event. As an example, consider an event following a 2-parameter Weibull distribution. The *cdf* of the 2-parameter Weibull distribution is given by:

$$F(T) = 1 - e^{-\left(\frac{T}{\eta}\right)^{\beta}}$$

The Weibull reliability function is given by:

$$\begin{aligned} R(T) &= 1 - F(t) \\ &= e^{-\left(\frac{T}{\eta}\right)^{\beta}} \end{aligned}$$

Then, to generate a random time from a Weibull distribution with a given $\eta$and β, a uniform random number from 0 to 1, $U_R[0,1]$, is first obtained. The random time from a Weibull distribution is then obtained from:

$$T_R = \eta \cdot \{-\ln\left[U_R[0,1]\right]\}^{\frac{1}{\beta}}$$

To obtain a conditional time, the Weibull conditional reliability function is given by:

$$R(T,t) = \frac{R(T+t)}{R(T)} = \frac{e^{-\left(\frac{T+t}{\eta}\right)^{\beta}}}{e^{-\left(\frac{T}{\eta}\right)^{\beta}}}$$

Or:

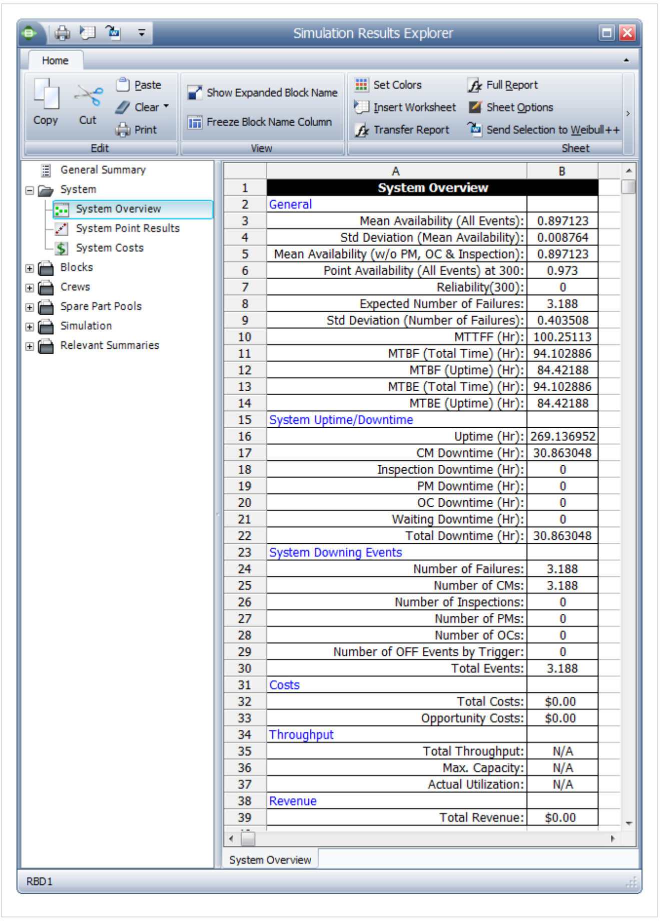
$$R(T,t) = e^{-\left[\left(\frac{T+t}{\eta}\right)^{\beta} - \left(\frac{T}{\eta}\right)^{\beta}\right]}$$

The random time would be the solution for $t$for $R(T,t) = U_R[0,1]$.

To illustrate the sequence of events, assume a single block with a failure and a repair distribution. The first event, E_{F1}, would be the failure of the component. Its first time-to-failure would be a random number drawn from its failure distribution, T_{F1}. Thus, the first failure event, E_{F1}, would be at T_{F1}. Once failed, the next event would be the repair of the component, E_{R1}. The time to repair the component would now be drawn from its repair distribution, T_{R1}. The component would be restored by time $T_{F1} + T_{R1}$. The next event would now be the second failure of the component after the repair, E_{F2}. This event would occur after a component operating time of T_{F2}after the item is restored (again drawn from the failure distribution), or at $T_{F1} + T_{R1} + T_{F2}$. This process is repeated until the end time. It is important to note that each run will yield a different sequence of events due to the probabilistic nature of the times. To arrive at the desired result, this process is repeated many times and the results from each run (simulation) are recorded. In other words, if we were to repeat this 1,000 times, we would obtain 1,000 different values for E_{F1}, or $\left[E_{F1_1}, E_{F1_2}, ..., E_{F1_{1,000}}\right]$. The average of these values, $\left(\frac{1}{1000}\sum_{i=1}^{1,000} E_{F1_i}\right)$, would then be the average time to the first event, E_{F1}, or the mean time to first failure (MTTFF) for the component. Obviously, if the component were to be 100% renewed after each repair, then this value would also be the same for the second failure, etc.

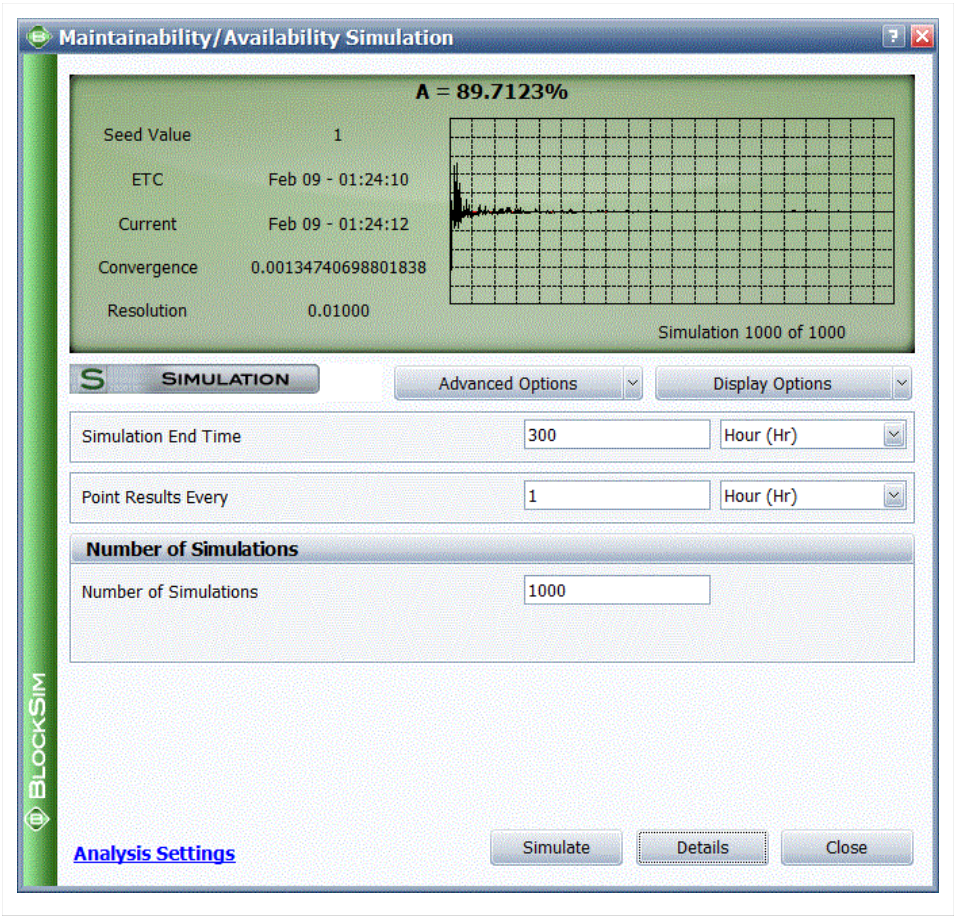
General Simulation Results

To further illustrate this, assume that components A and B in the prior example had normal failure and repair distributions with their means equal to the deterministic values used in the prior example and standard deviations of 10 and 1 respectively. That is, $F_A\tilde{}N(100,10)$, $F_B\tilde{}N(120,10)$, $R_A = R_B\tilde{}N(10,1)$. The settings for components C and D are not changed. Obviously, given the probabilistic nature of the example, the times to each event will vary. If one were to repeat this Xnumber of times, one would arrive at the results of interest for the system and its components. Some of the results for this system and this example, over 1,000 simulations, are provided in the figure below and explained in the next sections.



	A	B
1	System Overview	
2	General	
3	Mean Availability (All Events):	0.897123
4	Std Deviation (Mean Availability):	0.008764
5	Mean Availability (w/o PM, OC & Inspection):	0.897123
6	Point Availability (All Events) at 300:	0.973
7	Reliability(300):	0
8	Expected Number of Failures:	3.188
9	Std Deviation (Number of Failures):	0.403508
10	MTTFF (Hr):	100.25113
11	MTBF (Total Time) (Hr):	94.102886
12	MTBF (Uptime) (Hr):	84.42188
13	MTBE (Total Time) (Hr):	94.102886
14	MTBE (Uptime) (Hr):	84.42188
15	System Uptime/Downtime	
16	Uptime (Hr):	269.136952
17	CM Downtime (Hr):	30.863048
18	Inspection Downtime (Hr):	0
19	PM Downtime (Hr):	0
20	OC Downtime (Hr):	0
21	Waiting Downtime (Hr):	0
22	Total Downtime (Hr):	30.863048
23	System Downing Events	
24	Number of Failures:	3.188
25	Number of CMs:	3.188
26	Number of Inspections:	0
27	Number of PMs:	0
28	Number of OCs:	0
29	Number of OFF Events by Trigger:	0
30	Total Events:	3.188
31	Costs	
32	Total Costs:	$0.00
33	Opportunity Costs:	$0.00
34	Throughput	
35	Total Throughput:	N/A
36	Max. Capacity:	N/A
37	Actual Utilization:	N/A
38	Revenue	
39	Total Revenue:	$0.00

The simulation settings are shown in the figure below.


Maintainability/Availability Simulation

A = 89.7123%

Seed Value 1

ETC Feb 09 - 01:24:10

Current Feb 09 - 01:24:12

Convergence 0.00134740698801838

Resolution 0.01000

Simulation 1000 of 1000

Simulation End Time 300 Hour (Hr)

Point Results Every 1 Hour (Hr)

Number of Simulations

Number of Simulations 1000

General

Mean Availability (All Events), $\overline{A}_{ALL}$

This is the mean availability due to all downing events, which can be thought of as the operational availability. It is the ratio of the system uptime divided by the total simulation time (total time). For this example:

$$\begin{aligned}\overline{A}_{ALL} = & \frac{Uptime}{TotalTime} \\ = & \frac{269.137}{300} \\ = & 0.8971\end{aligned}$$

Std Deviation (Mean Availability)

This is the standard deviation of the mean availability of all downing events for the system during the simulation.

Mean Availability (w/o PM, OC & Inspection), $\overline{A}_{CM}$

This is the mean availability due to failure events only and it is 0.971 for this example. Note that for this case, the mean availability without preventive maintenance, on condition maintenance and inspection is identical to the mean availability for all events. This is because no preventive maintenance actions or inspections were defined for this system. We will discuss the inclusion of these actions in later sections.

Downtimes caused by PM and inspections are not included. However, if the PM or inspection action results in the discovery of a failure, then these times are included. As an example, consider a component that has failed but its failure is not discovered until the component is inspected. Then the downtime from the time failed to the time restored after the inspection is counted as failure downtime, since the original event that caused this was the component's failure.

Point Availability (All Events), $A\left(t\right)$

This is the probability that the system is up at time t. As an example, to obtain this value at t= 300, a special counter would need to be used during the simulation. This counter is increased by one every time the system is up at 300 hours. Thus, the point availability at 300 would be the times the system was up at 300 divided by the number of simulations. For this example, this is 0.930, or 930 times out of the 1000 simulations the system was up at 300 hours.

Reliability (Fail Events), $R(t)$

This is the probability that the system has not failed by time t. This is similar to point availability with the major exception that it only looks at the probability that the system did not have a single failure. Other (non-failure) downing events are ignored. During the simulation, a special counter again must be used. This counter is increased by one (once in each simulation) if the system has had at least one failure up to 300 hours. Thus, the reliability at 300 would be the number of times the system did not fail up to 300 divided by the number of simulations. For this example, this is 0 because the system failed prior to 300 hours 1000 times out of the 1000 simulations.

It is very important to note that this value is not always the same as the reliability computed using the analytical methods, depending on the redundancy present. The reason that it may differ is best explained by the following scenario:

Assume two units in parallel. The analytical system reliability, which does not account for repairs, is the probability that both units fail. In this case, when one unit goes down, it does not get repaired and the system fails after the second unit fails. In the case of repairs, however, it is possible for one of the two units to fail and get repaired before the second unit fails. Thus, when the second unit fails, the system will still be up due to the fact that the first unit was repaired.

Expected Number of Failures, N_F

This is the average number of system failures. The system failures (not downing events) for all simulations are counted and then averaged. For this case, this is 3.188, which implies that a total of 3,188 system failure events occurred over 1000 simulations. Thus, the expected number of system failures for one run is 3.188. This number includes all failures, even those that may have a duration of zero.

Std Deviation (Number of Failures)

This is the standard deviation of the number of failures for the system during the simulation.

MTTFF

MTTFF is the mean time to first failure for the system. This is computed by keeping track of the time at which the first system failure occurred for each simulation. MTTFF is then the average of these times. This may or may not be identical to the MTTF obtained in the analytical solution for the same reasons as those discussed in the Point Reliability section. For this case, this is 100.2511. This is fairly obvious for this case since the mean of one of the components in series was 100 hours.

It is important to note that for each simulation run, if a first failure time is observed, then this is recorded as the system time to first failure. If no failure is observed in the system, then the simulation end time is used as a right censored (suspended) data point. MTTFF is then computed using the total operating time until the first failure divided by the number of observed failures (constant failure rate assumption). Furthermore, and if the simulation end time is much less than the time to first failure for the system, it is also possible that all data points are right censored (i.e., no system failures were observed). In this case, the MTTFF is again computed using a constant failure rate assumption, or:

$$MTTFF = \frac{2\cdot (T_S)\cdot N}{\chi^2_{0.50;2}}$$

where $T_S$is the simulation end time and $N$is the number of simulations. One should be aware that this formulation may yield unrealistic (or erroneous) results if the system does not have a constant failure rate. If you are trying to obtain an accurate (realistic) estimate of this value, then your simulation end time should be set to a value that is well beyond the MTTF of the system (as computed analytically). As a general rule, the simulation end time should be at least three times larger than the MTTF of the system.

MTBF (Total Time)

This is the mean time between failures for the system based on the total simulation time and the expected number of system failures. For this example:

$$\begin{aligned} MTBF(TotalTime) = & \frac{TotalTime}{N_F} \\ = & \frac{300}{3.188} \\ = & 94.102886 \end{aligned}$$

MTBF (Uptime)

This is the mean time between failures for the system, considering only the time that the system was up. This is calculated by dividing system uptime by the expected number of system failures. You can also think of this as the mean uptime. For this example:

$$\begin{aligned} MTBF(Uptime) = & \frac{Uptime}{N_F} \\ = & \frac{269.136952}{3.188} \\ = & 84.42188 \end{aligned}$$

MTBE (Total Time)

This is the mean time between all downing events for the system, based on the total simulation time and including all system downing events. This is calculated by dividing the simulation run time by the number of downing events ($N_{ALL_{Down}}$).

MTBE (Uptime)

his is the mean time between all downing events for the system, considering only the time that the system was up. This is calculated by dividing system uptime by the number of downing events ($N_{ALL_{Down}}$).

System Uptime/Downtime

Uptime, T_{UP}

This is the average time the system was up and operating. This is obtained by taking the sum of the uptimes for each simulation and dividing it by the number of simulations. For this example, the uptime is 269.137. To compute the Operational Availability, A_o,for this system, then:

$$A_o = \frac{T_{UP}}{T_S}$$

CM Downtime, $T_{CM_{Down}}$

This is the average time the system was down for corrective maintenance actions (CM) only. This is obtained by taking the sum of the CM downtimes for each simulation and dividing it by the number of simulations. For this example, this is 30.863. To compute the Inherent Availability, A_I,for this system over the observed time (which may or may not be steady state, depending on the length of the simulation), then:

$$A_I = \frac{T_S - T_{CM_{Down}}}{T_S}$$

Inspection Downtime

This is the average time the system was down due to inspections. This is obtained by taking the sum of the inspection downtimes for each simulation and dividing it by the number of simulations. For this example, this is zero because no inspections were defined.

PM Downtime, $T_{PM_{Down}}$

This is the average time the system was down due to preventive maintenance (PM) actions. This is obtained by taking the sum of the PM downtimes for each simulation and dividing it by the number of simulations. For this example, this is zero because no PM actions were defined.

OC Downtime, $T_{OC_{Down}}$

This is the average time the system was down due to on-condition maintenance (PM) actions. This is obtained by taking the sum of the OC downtimes for each simulation and dividing it by the number of simulations. For this example, this is zero because no OC actions were defined.

Waiting Downtime, $T_{Wait_{Down}}$

This is the amount of time that the system was down due to crew and spare part wait times or crew conflict times. For this example, this is zero because no crews or spare part pools were defined.

Total Downtime, T_{Down}

This is the downtime due to all events. In general, one may look at this as the sum of the above downtimes. However, this is not always the case. It is possible to have actions that overlap each other, depending on the options and settings for the simulation. Furthermore, there are other events that can cause the system to go down that do not get counted in any of the above categories. As an example, in the case of standby redundancy with a switch delay, if the settings are to reactivate the failed component after repair, the system may be down during the switch-back action. This downtime does not fall into any of the above categories but it is counted in the total downtime.

For this example, this is identical to $T_{CM_{Down}}$.

System Downing Events

System downing events are events associated with downtime. Note that events with zero duration will appear in this section only if the task properties specify that the task brings the system down or if the task properties specify that the task brings the item down and the item's failure brings the system down.

Number of Failures, $N_{F_{Down}}$

This is the average number of system downing failures. Unlike the Expected Number of Failures, N_F,this number does not include failures with zero duration. For this example, this is 3.188.

Number of CMs, $N_{CM_{Down}}$

This is the number of corrective maintenance actions that caused the system to fail. It is obtained by taking the sum of all CM actions that caused the system to fail divided by the number of simulations. It does not include CM events of zero duration. For this example, this is 3.188. Note that this may differ from the Number of Failures, $N_{F_{Down}}$. An example would be a case where the system has failed, but due to other settings for the simulation, a CM is not initiated (e.g., an inspection is needed to initiate a CM).

Number of Inspections, $N_{I_{Down}}$

This is the number of inspection actions that caused the system to fail. It is obtained by taking the sum of all inspection actions that caused the system to fail divided by the number of simulations. It does not include inspection events of zero duration. For this example, this is zero.

Number of PMs, $N_{PM_{Down}}$

This is the number of PM actions that caused the system to fail. It is obtained by taking the sum of all PM actions that caused the system to fail divided by the number of simulations. It does not include PM events of zero duration. For this example, this is zero.

Number of OCs, $N_{OC_{Down}}$

This is the number of OC actions that caused the system to fail. It is obtained by taking the sum of all OC actions that caused the system to fail divided by the number of simulations. It does not include OC events of zero duration. For this example, this is zero.

Number of OFF Events by Trigger, $N_{OFF_{Down}}$

This is the total number of events where the system is turned off by state change triggers. An OFF event is not a system failure but it may be included in system reliability calculations. For this example, this is zero.

Total Events, $N_{ALL_{Down}}$

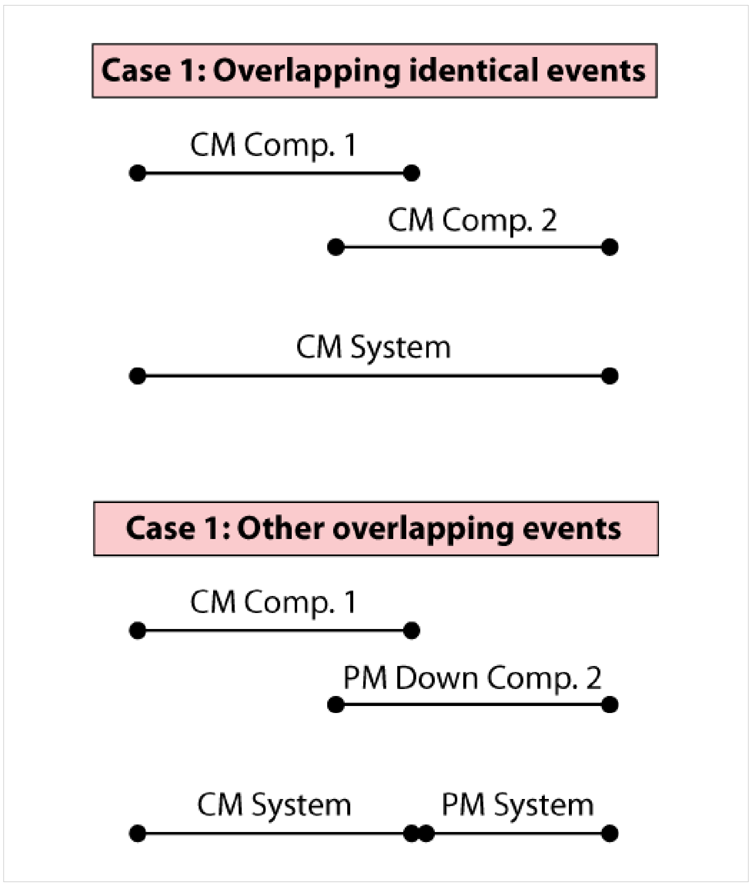
This is the total number of system downing events. It also does not include events of zero duration. It is possible that this number may differ from the sum of the other listed events. As an example, consider the case where a failure does not get repaired until an inspection, but the inspection occurs after the simulation end time. In this case, the number of inspections, CMs and PMs will be zero while the number of total events will be one.

Costs and Throughput

Cost and throughput results are discussed in later sections.

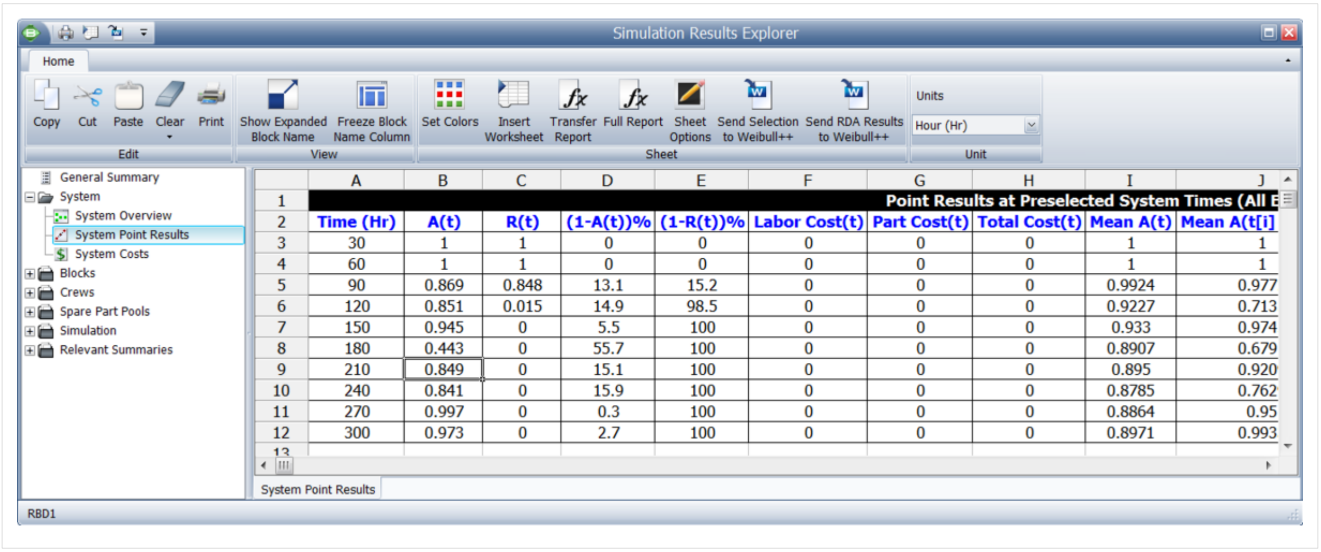
Note About Overlapping Downing Events

It is important to note that two identical system downing events (that are continuous or overlapping) may be counted and viewed differently. As shown in Case 1 of the following figure, two overlapping failure events are counted as only one event from the system perspective because the system was never restored and remained in the same down state, even though that state was caused by two different components. Thus, the number of downing events in this case is one and the duration is as shown in CM system. In the case that the events are different, as shown in Case 2 of the figure below, two events are counted, the CM and the PM. However, the downtime attributed to each event is different from the actual time of each event. In this case, the system was first down due to a CM and remained in a down state due to the CM until that action was over. However, immediately upon completion of that action, the system remained down but now due to a PM action. In this case, only the PM action portion that kept the system down is counted.



System Point Results

The system point results, as shown in the figure below, shows the Point Availability (All Events), $A\left(t\right)$, and Point Reliability, $R(t)$, as defined in the previous section. These are computed and returned at different points in time, based on the number of intervals selected by the user. Additionally, this window shows $(1-A(t))$, $(1-R(t))$, Labor Cost(t), Part Cost(t), $Cost(t)$, $Mean\ A(t)$, $Mean\ A(t_i - t_{i-1})$, $System$, $Failures(t)$, $\text{System Off Events by Trigger(t)}$and $Throughput(t)$.



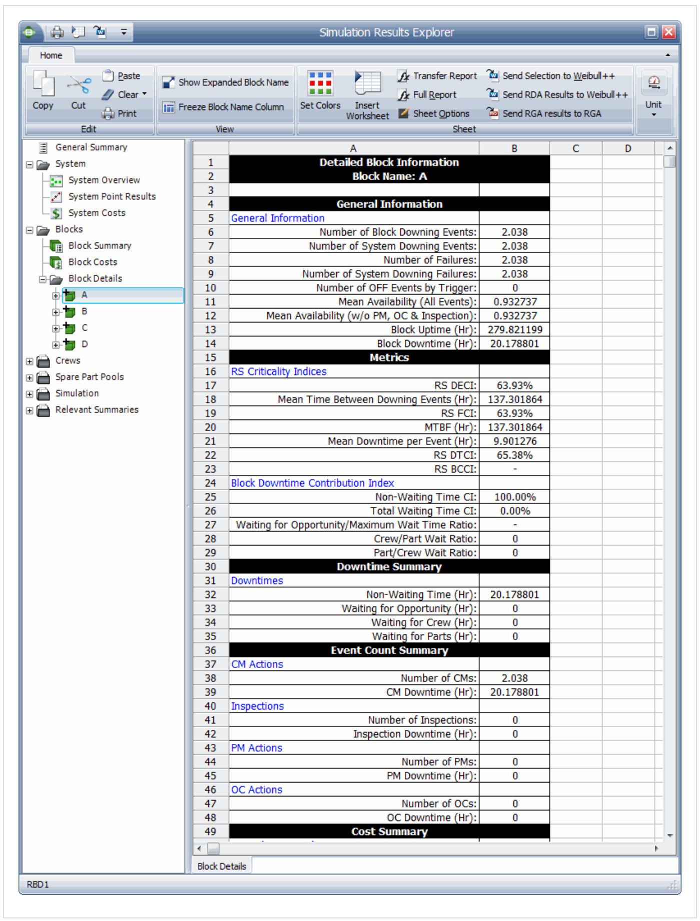
Point Results at Preselected System Times (All E

Time (Hr)	A(t)	R(t)	(1-A(t))%	(1-R(t))%	Labor Cost(t)	Part Cost(t)	Total Cost(t)	Mean A(t)	Mean A(t[i]
30	1	1	0	0	0	0	0	1	1
60	1	1	0	0	0	0	0	1	1
90	0.869	0.848	13.1	15.2	0	0	0	0.9924	0.977
120	0.851	0.015	14.9	98.5	0	0	0	0.9227	0.713
150	0.945	0	5.5	100	0	0	0	0.933	0.974
180	0.443	0	55.7	100	0	0	0	0.8907	0.679
210	0.849	0	15.1	100	0	0	0	0.895	0.920
240	0.841	0	15.9	100	0	0	0	0.8785	0.762
270	0.997	0	0.3	100	0	0	0	0.8864	0.95
300	0.973	0	2.7	100	0	0	0	0.8971	0.993

The number of intervals shown is based on the increments set. In this figure, the number of increments set was 300, which implies that the results should be shown every hour. The results shown in this figure are for 10 increments, or shown every 30 hours.

Results by Component

Simulation results for each component can also be viewed. The figure below shows the results for component A. These results are explained in the sections that follow.



Detailed Block Information **Block Name: A**	
General Information	
General Information	
Number of Block Downing Events:	2.038
Number of System Downing Events:	2.038
Number of Failures:	2.038
Number of System Downing Failures:	2.038
Number of OFF Events by Trigger:	0
Mean Availability (All Events):	0.932737
Mean Availability (w/o PM, OC & Inspection):	0.932737
Block Uptime (Hr):	279.821199
Block Downtime (Hr):	20.178801
Metrics	
RS Criticality Indices	
RS DECI:	63.93%
Mean Time Between Downing Events (Hr):	137.301864
RS FCI:	63.93%
MTBF (Hr):	137.301864
Mean Downtime per Event (Hr):	9.901276
RS DTCI:	65.38%
RS BCCI:	-
Block Downtime Contribution Index	
Non-Waiting Time CI:	100.00%
Total Waiting Time CI:	0.00%
Waiting for Opportunity/Maximum Wait Time Ratio:	-
Crew/Part Wait Ratio:	0
Part/Crew Wait Ratio:	0
Downtime Summary	
Downtimes	
Non-Waiting Time (Hr):	20.178801
Waiting for Opportunity (Hr):	0
Waiting for Crew (Hr):	0
Waiting for Parts (Hr):	0
Event Count Summary	
CM Actions	
Number of CMs:	2.038
CM Downtime (Hr):	20.178801
Inspections	
Number of Inspections:	0
Inspection Downtime (Hr):	0
PM Actions	
Number of PMs:	0
PM Downtime (Hr):	0
OC Actions	
Number of OCs:	0
OC Downtime (Hr):	0
Cost Summary	

General Information

Number of Block Downing Events, $Component_{NDE}$

This the number of times the component went down (failed). It includes all downing events.

Number of System Downing Events, $Component_{NSDE}$

This is the number of times that this component's downing caused the system to be down. For component A, this is 2.038. Note that this value is the same in this case as the number of component failures, since the component A is reliability-wise in series with components D and components B, C. If this were not the case (e.g., if they were in a parallel configuration, like B and C), this value would be different.

Number of Failures, $Component_{NF}$

This is the number of times the component failed and does not include other downing events. Note that this could also be interpreted as the number of spare parts required for CM actions for this component. For component A, this is 2.038.

Number of System Downing Failures, $Component_{NSDF}$

This is the number of times that this component's failure caused the system to be down. Note that this may be different from the Number of System Downing Events. It only counts the failure events that downed the system and does not include zero duration system failures.

Number of OFF events by Trigger, $Component_{OFF}$

The total number of events where the block is turned off by state change triggers. An OFF event is not a failure but it may be included in system reliability calculations.

Mean Availability (All Events), $\overline{A}_{ALL_{Component}}$

This has the same definition as for the system with the exception that this accounts only for the component.

Mean Availability (w/o PM, OC & Inspection), $\overline{A}_{CM_{Component}}$

The mean availability of all downing events for the block, not including preventive, on condition or inspection tasks, during the simulation.

Block Uptime, $T_{Component_{UP}}$

This is tThe total amount of time that the block was up (i.e., operational) during the simulation. For component A, this is 279.8212.

Block Downtime, $T_{Component_{Down}}$

This is the average time the component was down for any reason. For component A, this is 20.1788.

Block Downtime shows the total amount of time that the block was down (i.e., not operational) during the simulation.

Metrics

RS DECI

The ReliaSoft Downing Event Criticality Index for the block. This is a relative index showing the percentage of times that a downing event of the block caused the system to go down (i.e., the number of system downing events caused by the block divided by the total number of system downing events). For component A, this is 63.93%. This implies that 63.93% of the times that the system went down, the system failure was due to the fact that component A went down. This is obtained from:

$$RSDECI = \frac{Component_{NSDE}}{N_{ALL_{Down}}}$$

Mean Time Between Downing Events

This is the mean time between downing events of the component, which is computed from:

$$MTBDE = \frac{T_{Component_{UP}}}{Component_{NDE}}$$

For component A, this is 137.3019.

RS FCI

ReliaSoft's Failure Criticality Index (RS FCI) is a relative index showing the percentage of times that a failure of this component caused a system failure. For component A, this is 63.93%. This implies that 63.93% of the times that the system failed, it was due to the fact that component Afailed. This is obtained from:

$$RSFCI = \frac{Component_{NSDF} + F_{ZD}}{N_F}$$

$F_{ZD}$is a special counter of system failures not included in $Component_{NSDF}$. This counter is not explicitly shown in the results but is maintained by the software. The reason for this counter is the fact that zero duration failures are not counted in $Component_{NSDF}$since they really did not down the system. However, these zero duration failures need to be included when computing RS FCI.

It is important to note that for both RS DECI and RS FCI, and if overlapping events are present, the component that caused the system event gets credited with the system event. Subsequent component events that do not bring the system down (since the system is already down) do not get counted in this metric.

MTBF, $MTBF_C$

Mean time between failures is the mean (average) time between failures of this component, in real clock time. This is computed from:

$$MTBF_C = \frac{T_S - CFDowntime}{Component_{NF}}$$

$CFDowntime$is the downtime of the component due to failures only (without PM, OC and inspection). The discussion regarding what is a failure downtime that was presented in the section explaining Mean Availability (w/o PM & Inspection) also applies here. For component A, this is 137.3019. Note that this value could fluctuate for the same component depending on the simulation end time. As an example, consider the deterministic scenario for this component. It fails every 100 hours and takes 10 hours to repair. Thus, it would be failed at 100, repaired by 110, failed at 210 and repaired by 220. Therefore, its uptime is 280 with two failure events, MTBF = 280/2 = 140. Repeating the same scenario with an end time of 330 would yield failures at 100, 210 and 320. Thus, the uptime would be 300 with three failures, or MTBF = 300/3 = 100. Note that this is not the same as the MTTF (mean time to failure), commonly referred to as MTBF by many practitioners.

Mean Downtime per Event, $MDPE$

Mean downtime per event is the average downtime for a component event. This is computed from:

$$MDPE = \frac{T_{Component_{Down}}}{Component_{NDE}}$$

RS DTCI

The ReliaSoft Downtime Criticality Index for the block. This is a relative index showing the contribution of the block to the system's downtime (i.e., the system downtime caused by the block divided by the total system downtime).

RS BCCI

The ReliaSoft Block Cost Criticality Index for the block. This is a relative index showing the contribution of the block to the total costs (i.e., the total block costs divided by the total costs).

Non-Waiting Time CI

A relative index showing the contribution of repair times to the block's total downtime. (The ratio of the time that the crew is actively working on the item to the total down time).

Total Waiting Time CI

A relative index showing the contribution of wait factor times to the block's total downtime. Wait factors include crew conflict times, crew wait times and spare part wait times. (The ratio of downtime not including active repair time).

Waiting for Opportunity/Maximum Wait Time Ratio

A relative index showing the contribution of crew conflict times. This is the ratio of the time spent waiting for the crew to respond (not including crew logistic delays) to the total wait time (not including the active repair time).

Crew/Part Wait Ratio

The ratio of the crew and part delays. A value of 100% means that both waits are equal. A value greater than 100% indicates that the crew delay was in excess of the part delay. For example, a value of 200% would indicate that the wait for the crew is two times greater than the wait for the part.

Part/Crew Wait Ratio

The ratio of the part and crew delays. A value of 100% means that both waits are equal. A value greater than 100% indicates that the part delay was in excess of the crew delay. For example, a value of 200% would indicate that the wait for the part is two times greater than the wait for the crew.

Downtime Summary

Non-Waiting Time

Time that the block was undergoing active maintenance/inspection by a crew. If no crew is defined, then this will return zero.

Waiting for Opportunity

The total downtime for the block due to crew conflicts (i.e., time spent waiting for a crew while the crew is busy with another task). If no crew is defined, then this will return zero.

Waiting for Crew

The total downtime for the block due to crew wait times (i.e., time spent waiting for a crew due to logistical delay). If no crew is defined, then this will return zero.

Waiting for Parts

The total downtime for the block due to spare part wait times. If no spare part pool is defined then this will return zero.

Other Results of Interest

The remaining component (block) results are similar to those defined for the system with the exception that now they apply only to the component.

Imperfect Repairs

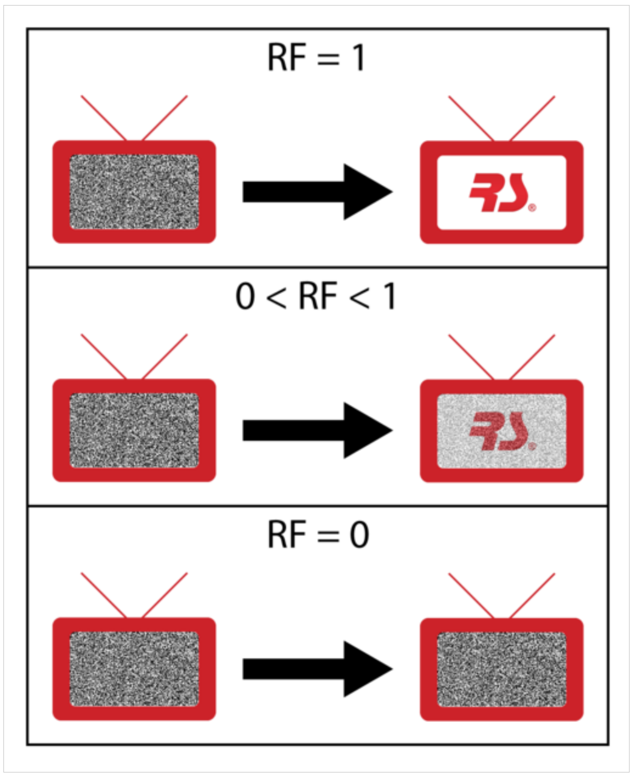
Restoration Factors (RF)

In the prior discussion it was assumed that a repaired component is as good as new after repair. This is usually the case when replacing a component with a new one. The concept of a restoration factor may be used in cases in which one wants to model imperfect repair, or a repair with a used component. The best way to indicate that a component is not as good as new is to give the component some age. As an example, if one is dealing with car tires, a tire that is not as good as new would have some pre-existing wear on it. In other words, the tire would have some accumulated mileage. A restoration factor concept is used to better describe the existing age of a component. The restoration factor is used to determine the age of the component after a repair or any other maintenance action (addressed in later sections, such as a PM action or inspection).

The restoration factor in BlockSim is defined as a number between 0 and 1 and has the following effect:

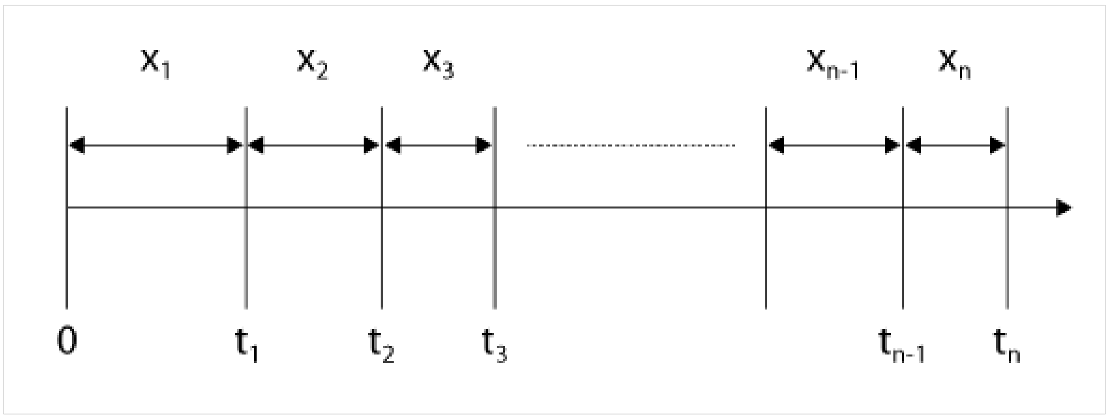
1. A restoration factor of 1 (100%) implies that the component is as good as new after repair, which in effect implies that the starting age of the component is 0.
2. A restoration factor of 0 implies that the component is the same as it was prior to repair, which in effect implies that the starting age of the component is the same as the age of the component at failure.
3. A restoration factor of 0.25 (25%) implies that the starting age of the component is equal to 75% of the age of the component at failure.

The figure below provides a visual demonstration of restoration factors. It should be noted that for successive maintenance actions on the same component, the age of the component after such an action is the initial age plus the time to failure since the last maintenance action.



Type I and Type II RFs

BlockSim offers two kinds of restoration factors. The type I restoration factor is based on Kijima [12, 13] model I and assumes that the repairs can only fix the wear-out and damage incurred during the last period of operation. Thus, the nth repair can only remove the damage incurred during the time between the (n-1)th and nth failures. The type II restoration factor, based on Kijima model II, assumes that the repairs fix all of the wear-out and damage accumulated up to the current time. As a result, the nth repair not only removes the damage incurred during the time between the (n-1)th and nth failures, but can also fix the cumulative damage incurred during the time from the first failure to the (n-1)th failure.



To illustrate this, consider a repairable system, observed from time $t = 0$, as shown in figure above. Let the successive failure times be denoted by t_1, t_2, ... and let the times between failures be denoted by x_1, x_2, ... Let RFdenote the restoration factor, then the age of the system $v_n$at time $t_n$using the two types of restoration factors is:

Type I Restoration Factor:

$$v_n = v_{n-1} + (1 - RF)x_n$$

Type II Restoration Factor:

$$v_n = (1 - RF)(v_{n-1} + x_n)$$

Illustrating Type I RF Through an Example

Assume that you have a component with a Weibull failure distribution ($\beta = 1.5$, $\eta = 1000$hours), RF type I = 0.25 and the component undergoes instant repair. Furthermore, assume that the component starts life new (i.e., with a start age of zero). The simulation steps are as follows:

1. Generate a uniform random number, $U_R[0,1] = 0.7021885$.
2. The first failure event will then be at 500 hours.
3. After instantaneous repair, the component will begin life with an age after repair of 350 hours $(500 \times (1 - 0.25))$.
4. Generate another uniform random number, $U_R[0,1] = 0.8824969$.
5. The next failure event is now determined using the conditional reliability equation, or:

$$\begin{aligned} R(t+T) =& R(t,T) \cdot R(T) \\ R(t+350) =& 0.8824969 \cdot R(350) \\ R(t+350) =& 0.8824969 \cdot 0.8129686 \\ R(t+350) =& 0.71744226 \\ t+350 =& 479.527 \\ t =& 129.527 \end{aligned}$$

Thus, the next failure event will be at $500 + 129.527 = 629.527$hours. Note that if the component had been as good as new (i.e., RF = 100%), then the next failure would have been at 750 hours (500 + 250), where 250 is the time corresponding to a reliability of 0.8824969, which is the random number that was generated in Step 4.

6. At this failure point, the item's age will now be equal to the initial age, after the first corrective action, plus the additional time it operated, or $350 + 129.527$hours.

7. Thus, the age after the second repair will be the sum of the previous age and the restoration factor times the age of the component since the last failure, or $350 + (129.527 \times (1 - 0.25)) = 447.14525$hours.

8. Go to Step 4 and repeat the process.

Illustrating Type II RF Through an Example

Assume that you have a component with a Weibull failure distribution ($\beta = 1.5$, $\eta = 1000\, hr$), RF type II = 0.25 and the component undergoes instant repair. Furthermore, assume that the component starts life new (i.e., with a start age of zero). The simulation steps are as follows:

1. Generate a uniform random number, $U_R[0,1] = 0.7021885$.
2. The first failure event will then be at 500 hrs.
3. After instantaneous repair, the component will begin life with an age after repair of 350 hrs $(500 \times (1 - 0.25))$.
4. Generate another uniform random number, $U_R[0,1] = 0.8824969$.
   1. The next failure event is now determined using the conditional reliability equation, or:

$$\begin{aligned} R(t+T) &= R(t,T)\cdot R(T) \\ R(t+350) &= 0.8824969\cdot R(350) \\ R(t+350) &= 0.8824969\cdot 0.8129686 \\ R(t+350) &= 0.71744226 \\ t+350 &= 479.527 \\ t &= 129.527 \end{aligned}$$

Thus, the next failure event will be at $500+129.527=629.527$hrs. Note that if the component had been as good as new (i.e., RF = 100%), then the next failure would have been at 750 hrs (500 + 250), where 250 is the time corresponding to a reliability of 0.8824969, which is the random number that was generated in Step 4.

6. At this failure point, the item's age will now be equal to the initial age, after the first corrective action, plus the additional time it operated, or $350+129.527$.

7. Thus, the age after the second repair will be the restoration factor times the age of the component at failure, or $(350+129.527)\times(1-0.25)=359.64525$hrs.

8. Go to Step 4 and repeat the process.

Discussion of Type I and Type II RFs

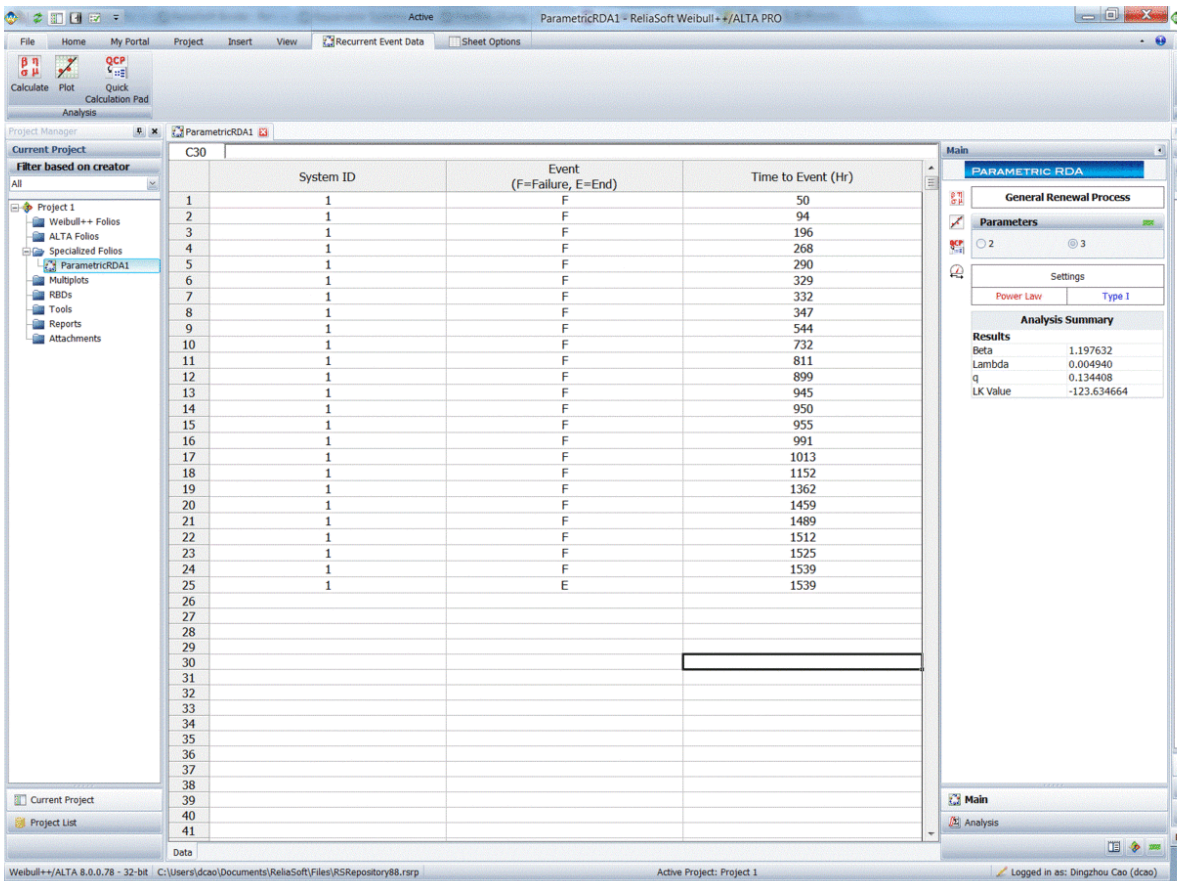
As an application example, consider an automotive engine that fails after six years of operation. The engine is rebuilt. The rebuild has the effect of rejuvenating the engine to a condition as if it were three years old (i.e., a 50% RF). Assume that the rebuild affects all of the damage on the engine (i.e., a Type II restoration). The engine fails again after three years (when it again reaches an age of six) and another rebuild is required. This rebuild will also rejuvenate the engine by 50%, thus making it three years old again.

Now consider a similar engine subjected to a similar rebuild, but that the rebuild only affects the damage since the last repair (i.e., a Type I restoration of 50%). The first rebuild will rejuvenate the engine to a three-year-old condition. The engine will fail again after three years, but the rebuild this time will only affect the age (of three years) after the first rebuild. Thus the engine will have an age of four and a half years after the second rebuild ($3+3\times(1-0.5)=4.5$). After the second rebuild the engine will fail again after a period of one and a half years and a third rebuild will be required. The age of the engine after the third rebuild will be five years and three months ($4.5+1.5\times(1-0.5)=5.25$).

It should be pointed out that when dealing with constant failure rates (i.e., with a distribution such as the exponential), the restoration factor has no effect.

Calculations to Obtain RFs

The two types of restoration factors discussed in the previous sections can be calculated using the parametric RDA (Recurrent Data Analysis) tool in Weibull++ 8. This tool uses the GRP (General Renewal Process) model to analyze failure data of a repairable item. More information on the Parametric RDA tool and the GRP (General Renewal Process) model can be found in [25]. As an example, consider the times to failure for an air-conditioning unit of an aircraft recorded in the following table. Assume that each time the unit is repaired, the repair can only remove the damage incurred during the last period of operation. This assumption implies a type I RF factor which is specified as an analysis setting in the Weibull++ folio. The type I RF for the air-conditioning unit can be calculated using the results from Weibull++ shown in the figure below.



	System ID	Event (F=Failure, E=End)	Time to Event (Hr)
1	1	F	50
2	1	F	94
3	1	F	196
4	1	F	268
5	1	F	290
6	1	F	329
7	1	F	332
8	1	F	347
9	1	F	544
10	1	F	732
11	1	F	811
12	1	F	899
13	1	F	945
14	1	F	950
15	1	F	955
16	1	F	991
17	1	F	1013
18	1	F	1152
19	1	F	1362
20	1	F	1459
21	1	F	1489
22	1	F	1512
23	1	F	1525
24	1	F	1539
25	1	E	1539

PARAMETRIC RDA

General Renewal Process

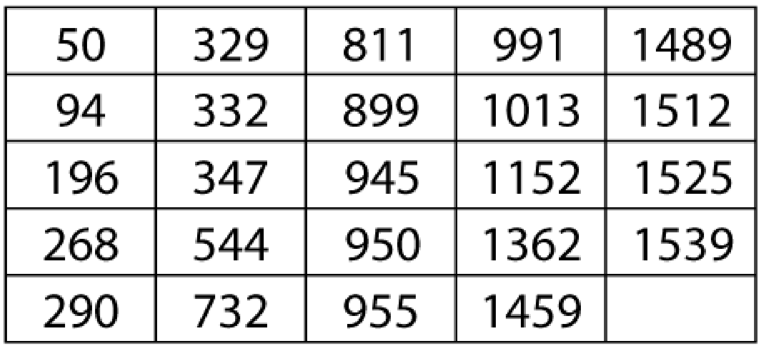
Parameters: 2 [✓ 3]

Settings: Power Law, Type I

Analysis Summary

Results	
Beta	1.197632
Lambda	0.004940
q	0.134408
LK Value	-123.634664

Using the Parametric RDA tool in Weibull++ to calculate restoration factors.



50	329	811	991	1489
94	332	899	1013	1512
196	347	945	1152	1525
268	544	950	1362	1539
290	732	955	1459	

The value of the action effectiveness factor q obtained from Weibull++ is:

$$q = 0.1344$$

The type I RF factor is calculated using q as:

$$\begin{aligned} RF &= 1 - q \\ &= 1 - 0.1344 \\ &= 0.8656 \end{aligned}$$

The parameters of the Weibull distribution for the air-conditioning unit can also be calculated. β is obtained from Weibull++ as 1.1976. η can be calculated using the β and λ values from Weibull++ as:

$$\begin{aligned}\eta &= \left(\frac{1}{\lambda}\right)^{\frac{1}{\beta}} \\ &= \left(\frac{1}{0.0049}\right)^{\frac{1}{1.1976}} \\ &= 84.8582\end{aligned}$$

The values of the type I RF, $\beta$and ηcalculated above can now be used to model the air-conditioning unit as a component in BlockSim.

Using Resources: Pools and Crews

In order to make the analysis more realistic, one may wish to consider additional sources of delay times in the analysis or study the effect of limited resources. In the prior examples, we used a repair distribution to identify how long it takes to restore a component. The factors that one chooses to consider in this time may include the time it takes to do the repair and/or the time it takes to get a crew, a spare part, etc. While all of these factors may be included in the repair duration, optimized usage of these resources can only be achieved if the resources are studied individually and their dependencies are identified.

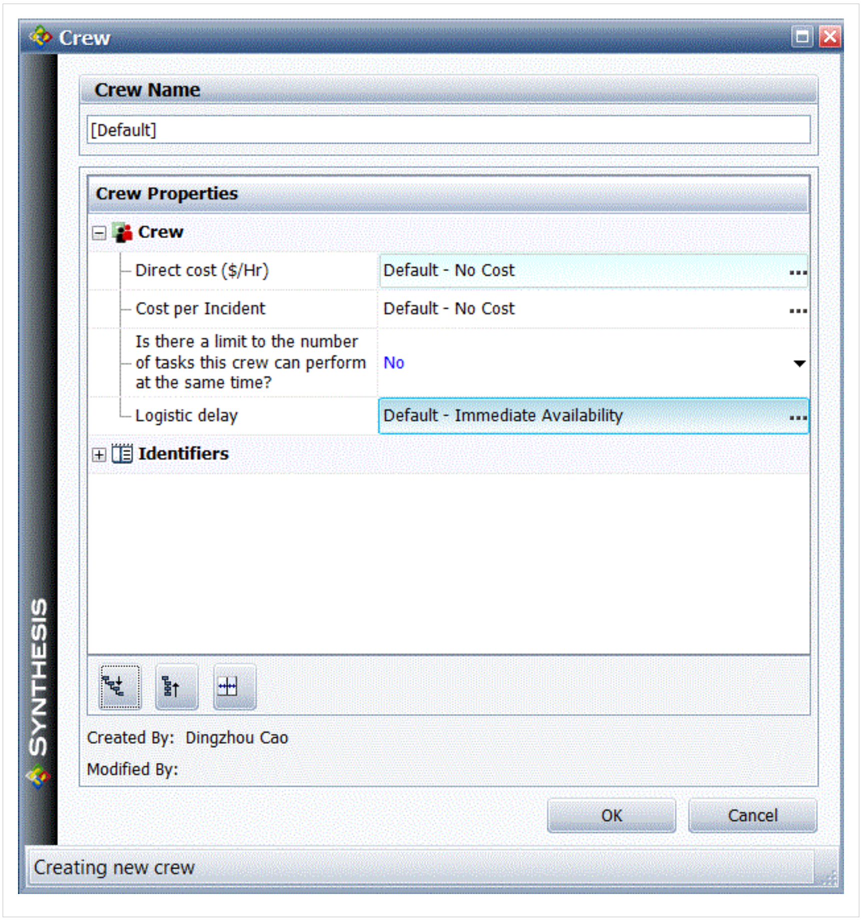
As an example, consider the situation where two components in parallel fail at the same time and only a single repair person is available. Because this person would not be able to execute the repair on both components simultaneously, an additional delay will be encountered that also needs to be included in the modeling. One way to accomplish this is to assign a specific repair crew to each component.

Including Crews

BlockSim allows you to assign maintenance crews to each component and one or more crews may be assigned to each component from the Maintenance Task Properties window. Note that there may be different crews for each action, (i.e., corrective, preventive, on condition and inspection).

A crew record needs to be defined for each named crew, as shown in the picture below. The basic properties for each crew include factors such as:

- Logistic delays. How long does it take for the crew to arrive?
- Is there a limit to the number of tasks this crew can perform at the same time? If yes, how many simultaneous tasks can the crew perform?
- What is the cost per hour for the crew?
- What is the cost per incident for the crew?


Crew

Crew Name

[Default]

Crew Properties

Crew

Direct cost ($/Hr)	Default - No Cost
Cost per Incident	Default - No Cost
Is there a limit to the number of tasks this crew can perform at the same time?	No
Logistic delay	Default - Immediate Availability

Identifiers

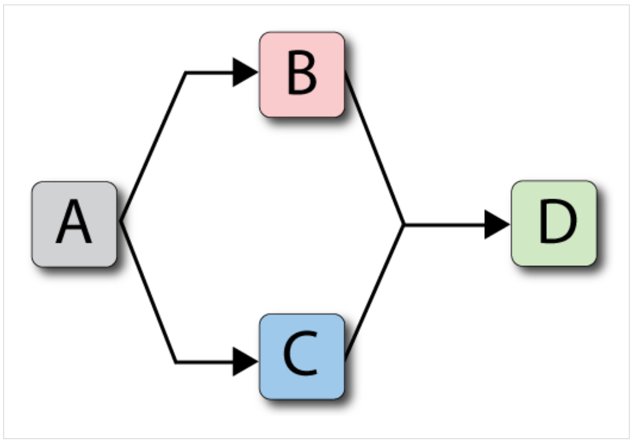
Created By: Dingzhou Cao

Modified By:

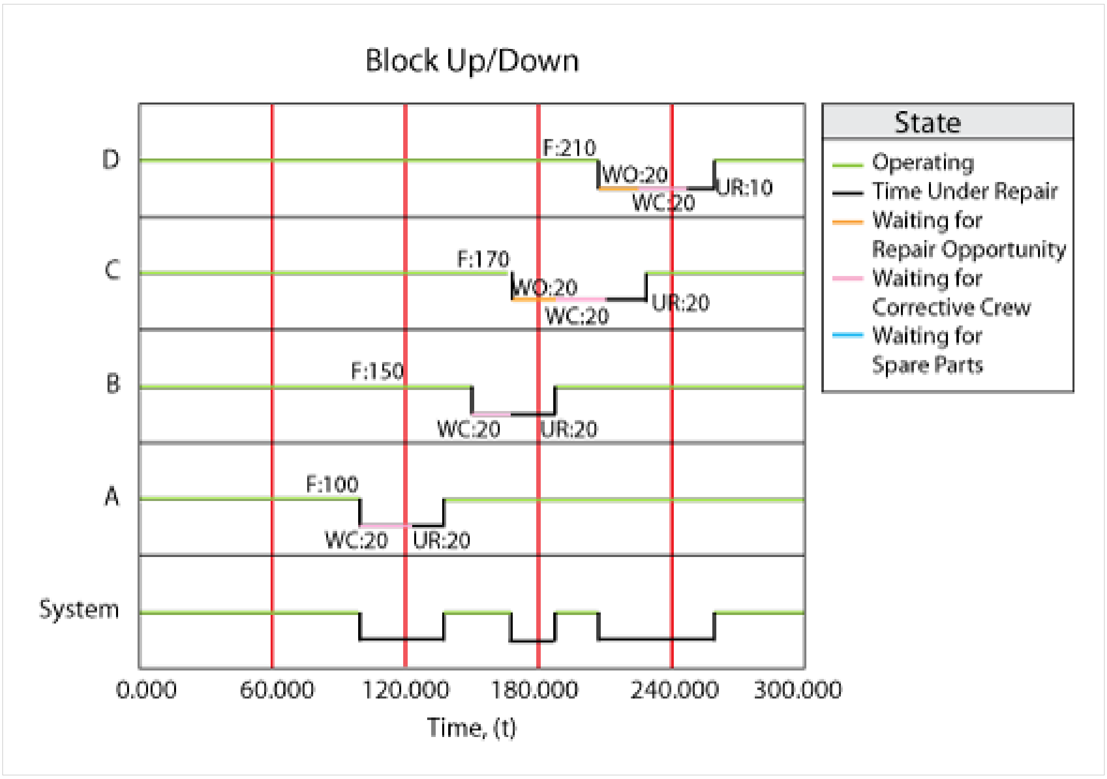
Creating new crew

Illustrating Crew Use

To illustrate the use of crews in BlockSim, consider the deterministic scenario described by the following RBD and properties.



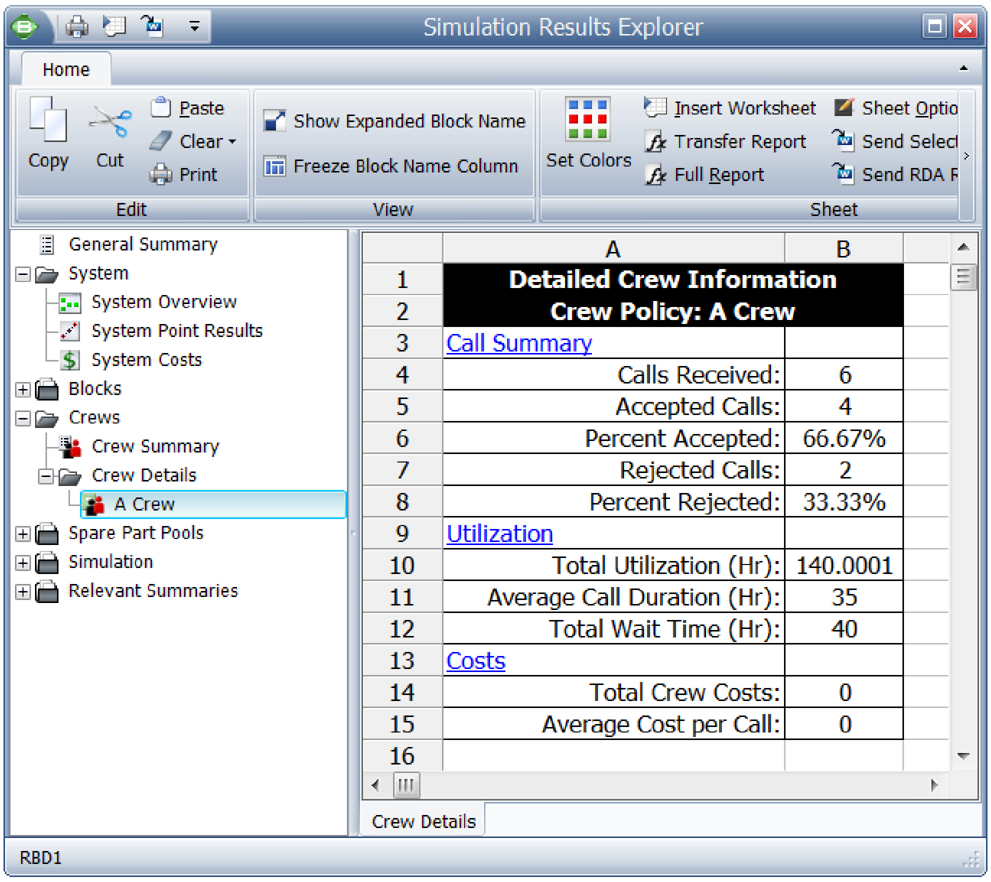
Unit	Failure	Repair	Crew
A	100	10	Crew A: Delay = 20, Single Task
B	120	20	Crew A: Delay = 20, Single Task
C	140	20	Crew A: Delay = 20, Single Task
D	160	10	Crew A: Delay = 20, Single Task



As shown in the figure above, the System Up/Down plot illustrates the sequence of events, which are:

1. At 100, Afails. It takes 20 to get the crew and 10 to repair, thus the component is repaired by 130. The system is failed/down during this time.
2. At 150, Bfails since it would have accumulated an operating age of 120 by this time. It again has to wait for the crew and is repaired by 190.
3. At 170, Cfails. Upon this failure, Crequests the only available crew. However, this crew is currently engaged by $B$and, since the crew can only perform one task at a time, it cannot respond immediately to the request by C. Thus, Cwill remain failed until the crew becomes available. The crew will finish with unit Bat 190 and will then be dispatched to C. Upon dispatch, the logistic delay will again be considered and Cwill be repaired by 230. The system continues to operate until the failures of $B$and Coverlap (i.e., the system is down from 170 to 190)
4. At 210, Dfails. It again has to wait for the crew and repair.
5. $D$is up at 260.

The following figure shows an example of some of the possible crew results (details), which are presented next.



	A	B
1	Detailed Crew Information	
2	Crew Policy: A Crew	
3	Call Summary	
4	Calls Received:	6
5	Accepted Calls:	4
6	Percent Accepted:	66.67%
7	Rejected Calls:	2
8	Percent Rejected:	33.33%
9	Utilization	
10	Total Utilization (Hr):	140.0001
11	Average Call Duration (Hr):	35
12	Total Wait Time (Hr):	40
13	Costs	
14	Total Crew Costs:	0
15	Average Cost per Call:	0
16		

Crew results shown in the BlockSim's Simulation Results Explorer.

Explanation of the Crew Details

1. Each request made to a crew is logged.
2. If a request is successful (i.e., the crew is available), the call is logged once in the Calls Received counter and once in the Accepted Calls counter.
3. If a request is not accepted (i.e., the crew is busy), the call is logged once in the Calls Received counter and once in the Rejected Calls counter. When the crew is free and can be called upon again, the call is logged once in the Calls Received counter and once in the Accepted Calls counter.
4. In this scenario, there were two instances when the crew was not available, Rejected Calls = 2, and there were four instances when the crew performed an action, Calls Accepted = 4, for a total of six calls, Calls Received = 6.
5. Percent Accepted and Percent Rejected are the ratios of calls accepted and calls rejected with respect to the total calls received.
6. Total Utilization is the total time that the crew was used. It includes both the time required to complete the repair action and the logistic time. In this case, this is 140, or:

$$\begin{aligned}
T_{R_A} &= 10, T_{L_A} = 20 \\
T_{R_B} &= 20, T_{L_B} = 20 \\
T_{R_C} &= 20, T_{L_C} = 20 \\
T_{R_D} &= 10, T_{L_D} = 20 \\
T_U &= (T_{R_A} + T_{L_A}) + (T_{R_B} + T_{L_B}) \\
&\quad + (T_{R_C} + T_{L_C}) + (T_{R_D} + T_{L_D}) \\
T_U &= 140
\end{aligned}$$

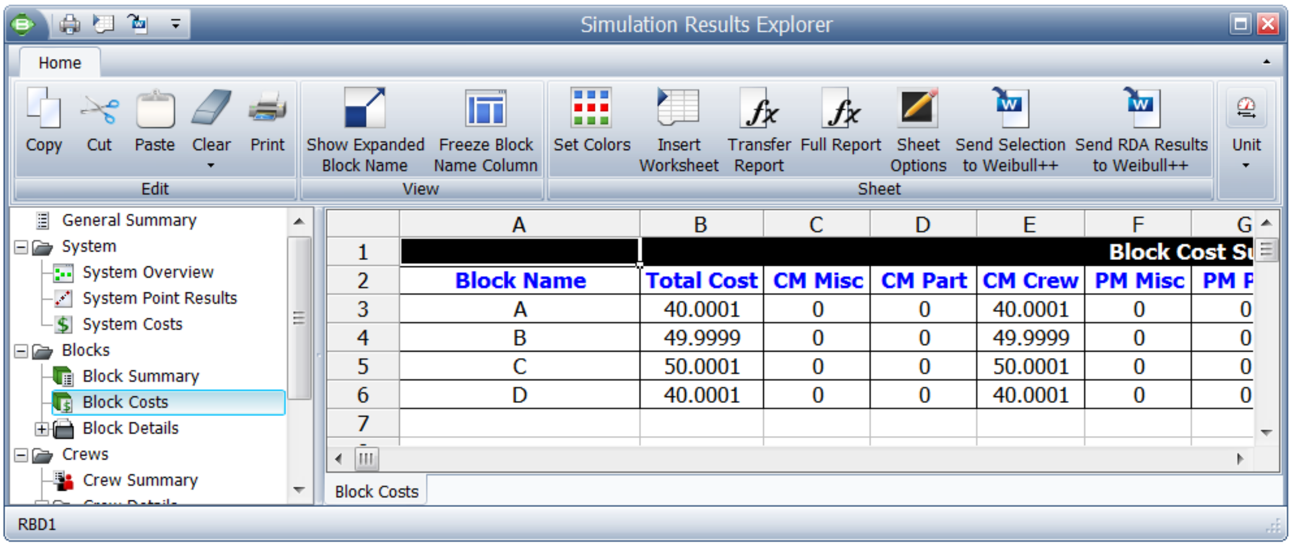
6. Average Call Duration is the average duration of each crew usage, and it also includes both logistic and repair time. It is the total usage divided by the number of accepted calls. In this case, this is 35.

7. Total Wait Time is the time that blocks in need of a repair waited for this crew. In this case, it is 40 ($C$and Dboth waited 20 each).

8. Total Crew Costs are the total costs for this crew. It includes the per incident charge as well as the per unit time costs. In this case, this is 180. There were four incidents at 10 each for a total of 40, as well as 140 time units of usage at 1 cost unit per time unit.

9. Average Cost per Call is the total cost divided by the number of accepted calls. In this case, this is 45.

Note that crew costs that are attributed to individual blocks can be obtained from the Blocks reports, as shown in the figure below.



	A	B	C	D	E	F	G
1		Block Cost St					
2	Block Name	Total Cost	CM Misc	CM Part	CM Crew	PM Misc	PM P
3	A	40.0001	0	0	40.0001	0	0
4	B	49.9999	0	0	49.9999	0	0
5	C	50.0001	0	0	50.0001	0	0
6	D	40.0001	0	0	40.0001	0	0
7							

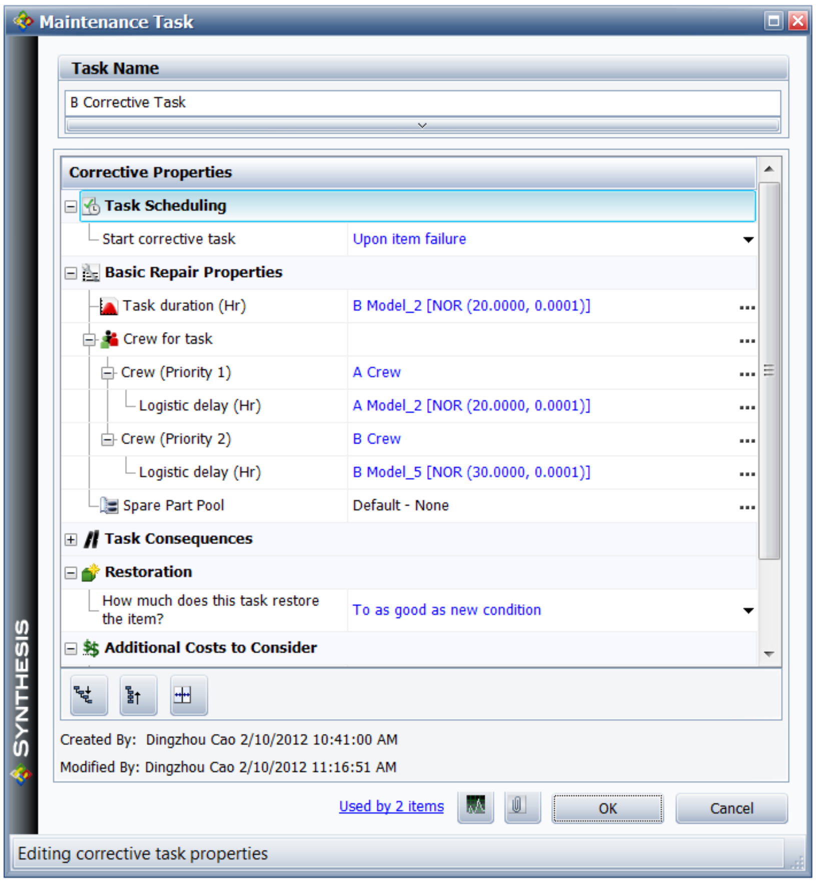
Allocation of crew costs.

How BlockSim Handles Crews

1. Crew logistic time is added to each repair time.
2. The logistic time is always present, and the same, regardless of where the crew was called from (i.e., whether the crew was at another job or idle at the time of the request).
3. For any given simulation, each crew's logistic time is constant (taken from the distribution) across that single simulation run regardless of the task (CM, PM or inspection).
4. A crew can perform either a finite number of simultaneous tasks or an infinite number.
5. If the finite limit of tasks is reached, the crew will not respond to any additional request until the number of tasks the crew is performing is less than its finite limit.
6. If a crew is not available to respond, the component will "wait" until a crew becomes available.
7. BlockSim maintains the queue of rejected calls and will dispatch the crew to the next repair on a "first come, first served" basis.
8. Multiple crews can be assigned to a single block (see overview in the next section).
9. If no crew has been assigned for a block, it is assumed that no crew restrictions exist and a default crew is used. The default crew can perform an infinite number of simultaneous tasks and has no delays or costs.

Looking at Multiple Crews

Multiple crews may be available to perform maintenance for a particular component. When multiple crews have been assigned to a block in BlockSim, the crews are assigned to perform maintenance based on their order in the crew list, as shown in the figure below.



Task Name

B Corrective Task

Corrective Properties

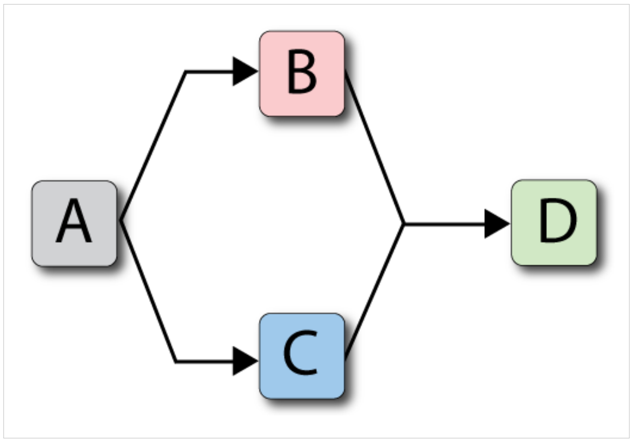
Task Scheduling	
Start corrective task	Upon item failure
Basic Repair Properties	
Task duration (Hr)	B Model_2 [NOR (20.0000, 0.0001)]
Crew for task	
Crew (Priority 1)	A Crew
Logistic delay (Hr)	A Model_2 [NOR (20.0000, 0.0001)]
Crew (Priority 2)	B Crew
Logistic delay (Hr)	B Model_5 [NOR (30.0000, 0.0001)]
Spare Part Pool	Default - None
Task Consequences	
Restoration	
How much does this task restore the item?	To as good as new condition
Additional Costs to Consider	

Created By: Dingzhou Cao 2/10/2012 10:41:00 AM
Modified By: Dingzhou Cao 2/10/2012 11:16:51 AM

Editing corrective task properties

A single component with two corrective maintenance crews assigned to it.

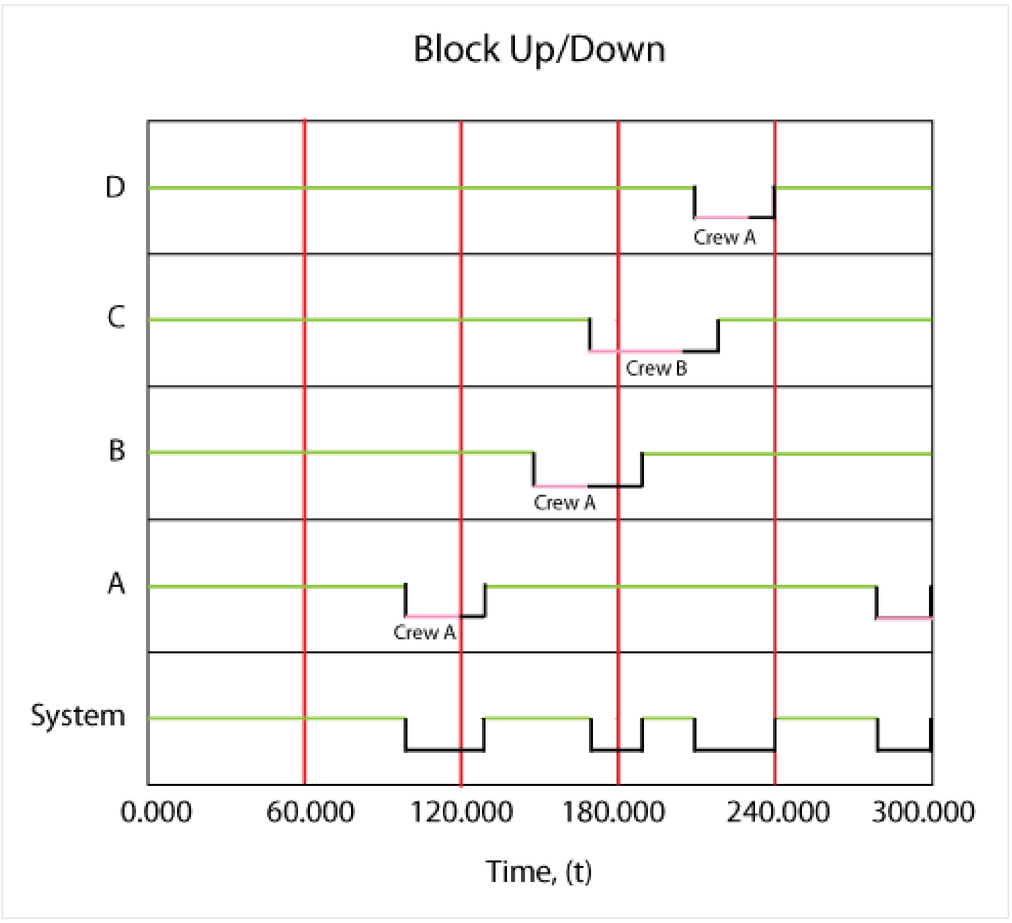
In the case where more than one crew is assigned to a block, and if the first crew is unavailable, then the next crew is called upon and so forth. As an example, consider the prior case but with the following modifications (i.e., Crews A and $B$are assigned to all blocks):



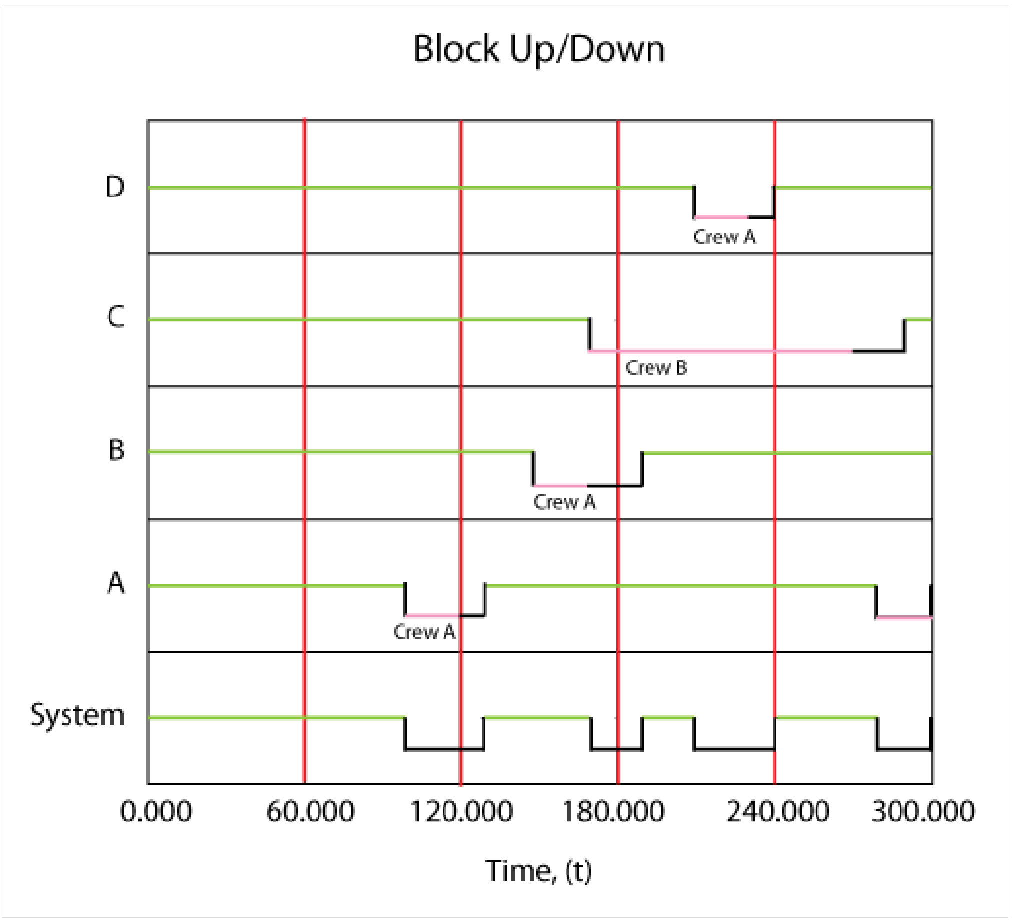
Unit	**Failure**	**Repair**	**Crew**
A	100	10	A, B
B	120	20	A, B
C	140	20	A, B
D	160	10	A, B

Crew A; Delay = 20, Single Task
Crew B; Delay = 30, Single Task

The system would behave as shown in the figure below.



In this case, Crew $B$was used for the Crepair since Crew $A$was busy. On all others, Crew $A$was used. It is very important to note that once a crew has been assigned to a task it will complete the task. For example, if we were to change the delay time for Crew $B$to 100, the system behavior would be as shown in the figure below.



In other words, even though Crew Awould have finished the repair on Cmore quickly if it had been available when originally called, $B$was assigned the task because $A$was not available at the instant that the crew was needed.

Additional Rules on Crews

1. If all assigned crews are engaged, the next crew that will be chosen is the crew that can get there first.

   a) This accounts for the time it would take a particular crew to complete its current task (or all tasks in its queue) and its logistic time.

2. If a crew is available, it gets used regardless of what its logistic delay time is.

   a) In other words, if a crew with a shorter logistic time is busy, but almost done, and another crew with a much higher logistic time is currently free, the free one will get assigned to the task.

3. For each simulation each crew's logistic time is computed (taken randomly from its distribution or its fixed time) at the beginning of the simulation and remains constant across that one simulation for all actions (CM, PM and inspection).

Using Spare Part Pools

BlockSim also allows you to specify spare part pools (or depots). Spare part pools allow you to model and manage spare part inventory and study the effects associated with limited inventories. Each component can have a spare part pool associated with it. If a spare part pool has not been defined for a block, BlockSim's analysis assumes a default pool of infinite spare parts. To speed up the simulation, no details on pool actions are kept during the simulation if the default pool is used.

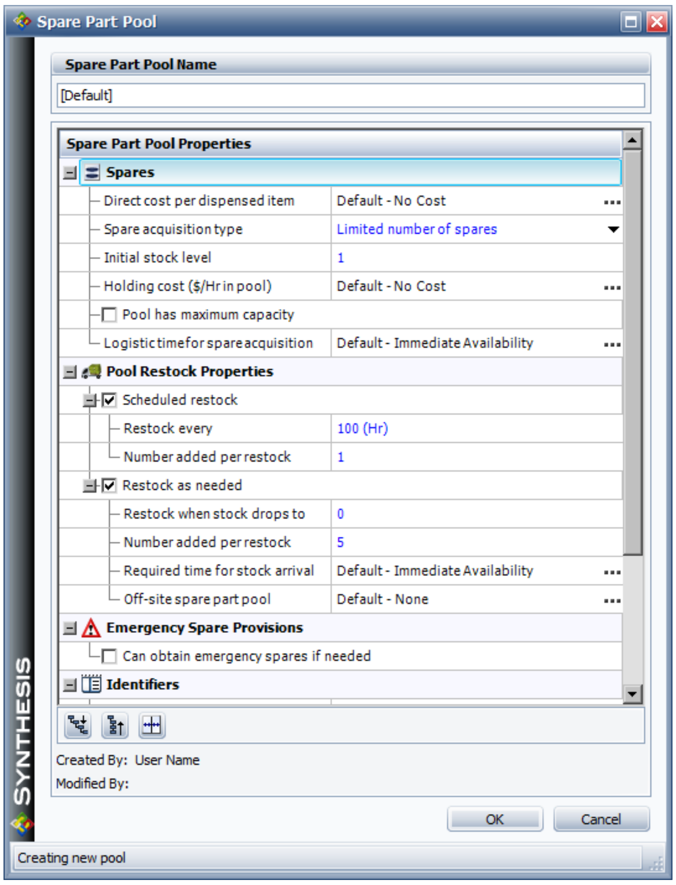
Pools allow you to define multiple aspects of the spare part process, including stock levels, logistic delays and restock options. Every time a part is repaired under a CM or scheduled action (PM, OC and Inspection), a spare part is obtained from the pool. If a part is available in the pool, it is then used for the repair. Spare part pools perform their actions based on the simulation clock time.

Spare Properties

A spare part pool is identified by a name. The general properties of the pool are its stock level (must be greater than zero), cost properties and logistic delay time. If a part is available (in stock), the pool will dispense that part to the requesting block after the specified logistic time has elapsed. One needs to think of a pool as an independent entity. It accepts requests for parts from blocks and dispenses them to the requesting blocks after a given logistic time. Requests for spares are handled on a first come, first served basis. In other words, if two blocks request a part and only one part is in stock, the first block that made the request will receive the part. Blocks request parts from the pool immediately upon the initiation of a CM or scheduled event (PM, OC and Inspection).

Restocking the Pool

If the pool has a finite number of spares, restock actions may be incorporated. The figure below shows the restock properties. Specifically, a pool can restock itself either through a scheduled restock action or based on specified conditions.


Spare Part Pool

Spare Part Pool Name

[Default]

Spare Part Pool Properties

Spares	
Direct cost per dispensed item	Default - No Cost
Spare acquisition type	Limited number of spares
Initial stock level	1
Holding cost ($/Hr in pool)	Default - No Cost
☐ Pool has maximum capacity	
Logistic time for spare acquisition	Default - Immediate Availability
Pool Restock Properties	
☑ Scheduled restock	
Restock every	100 (Hr)
Number added per restock	1
☑ Restock as needed	
Restock when stock drops to	0
Number added per restock	5
Required time for stock arrival	Default - Immediate Availability
Off-site spare part pool	Default - None
Emergency Spare Provisions	
☐ Can obtain emergency spares if needed	
Identifiers	

Created By: User Name
Modified By:

Creating new pool

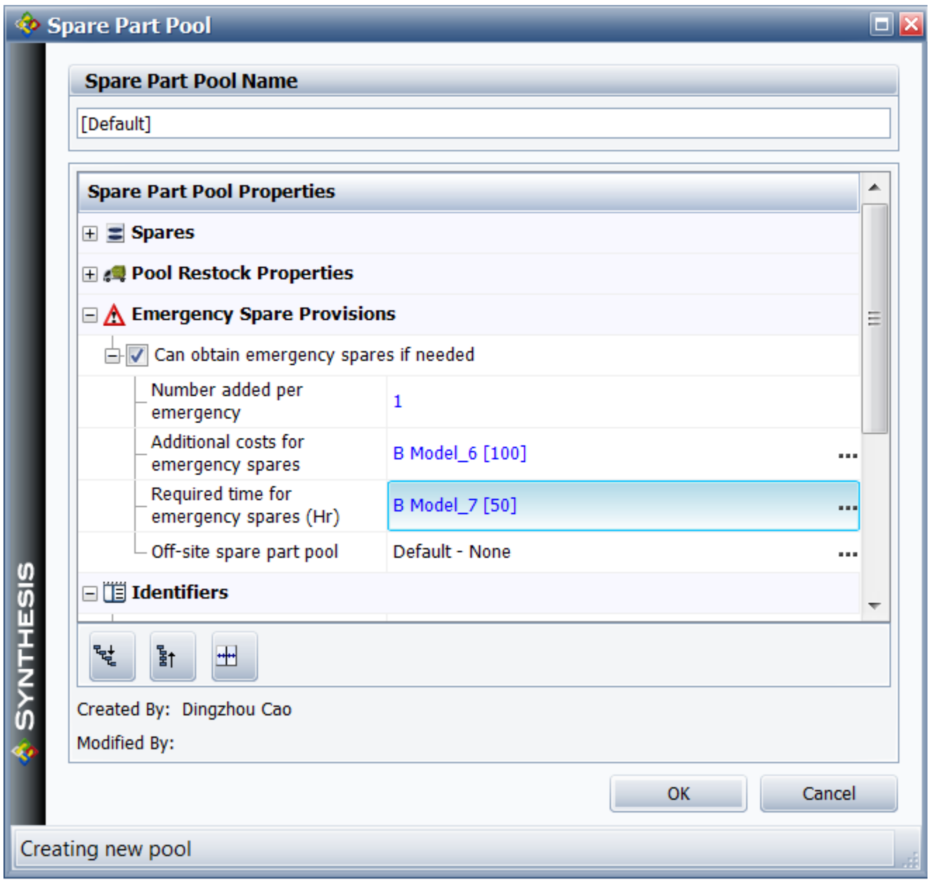
A scheduled restock action adds a set number of parts to the pool on a predefined scheduled part arrival time. For the settings in the figure above, one spare part would be added to the pool every 100 hours, based on the system (simulation) time. In other words, for a simulation of 1,000 hours, a spare part would arrive at 100 hours, 200 hours, etc. The part is available to the pool immediately after the restock action and without any logistic delays.

In an on-condition restock, a restock action is initiated when the stock level reaches (or is below) a specified value. In figure above, five parts are ordered when the stock level reaches 0. Note that unlike the scheduled restock, parts added through on-condition restock become available after a specified logistic delay time. In other words, when doing a scheduled restock, the parts are pre-ordered and arrive when needed. Whereas in the on-condition restock, the parts are ordered when the condition occurs and thus arrive after a specified time. For on-condition restocks, the condition is triggered if and only if the stock level drops to or below the specified stock level, regardless of how the spares arrived to the pool or were distributed by the pool. In addition, the restock trigger value must be less than the initial stock.

Lastly, a maximum capacity can be assigned to the pool. If the maximum capacity is reached, no more restock actions are performed. This maximum capacity must be equal to or greater than the initial stock. When this limit is reached, no more items are added to the pool. For example, if the pool has a maximum capacity of ten and a current stock level of eight and if a restock action is set to add five items to the pool, then only two will be accepted.

Obtaining Emergency Spares

Emergency restock actions can also be defined. The figure below illustrates BlockSim's Emergency Spare Provisions options. An emergency action is triggered only when a block requests a spare and the part is not currently in stock. This is the only trigger condition. It does not account for whether a part has been ordered or if one is scheduled to arrive. Emergency spares are ordered when the condition is triggered and arrive after a time equal to the required time to obtain emergency spare(s).



Spare Part Pool Name

[Default]

Spare Part Pool Properties

Spares

Pool Restock Properties

Emergency Spare Provisions

✓ Can obtain emergency spares if needed

Number added per emergency	1
Additional costs for emergency spares	B Model_6 [100]
Required time for emergency spares (Hr)	B Model_7 [50]
Off-site spare part pool	Default - None

Identifiers

Created By: Dingzhou Cao

Modified By:

Creating new pool

Summary of Rules for Spare Part Pools

The following rules summarize some of the logic when dealing with spare part pools.

Basic Logic Rules

1. **Queue Based**: Requests for spare parts from blocks are queued and executed on a "first come, first served" basis.

2. **Emergency**: Emergency restock actions are performed only when a part is not available.

3. **Scheduled Restocks**: Scheduled restocks are added instantaneously to the pool at the scheduled time.

4. **On-Condition Restock**: On-condition restock happens when the specified condition is reached (e.g., when the stock drops to two or if a request is received for a part and the stock is below the restock level).

   a) For example, if a pool has three items in stock and it dispenses one, an on-condition restock is initiated the instant that the request is received (without regard to the logistic delay time). The restocked items will be available after the required time for stock arrival has elapsed.

   b) The way that this is defined allows for the possibility of multiple restocks. Specifically, every time a part needs to be dispensed and the stock is lower than the specified quantity, parts are ordered. In the case of a long logistic delay time, it is possible to have multiple re-orders in the queue.

5. **Parts Become Available after Spare Acquisition Logistic Delay**: If there is a spare acquisition logistic time delay, the requesting block will get the part after that delay.

   a) For example, if a block with a repair duration of 10 fails at 100 and requests a part from a pool with a logistic delay time of 10, that block will not be up until 120.

6. **Compound Delays**: If a part is not available and an emergency part (or another part) can be obtained, then the total wait time for the part is the sum of both the logistic time and the required time to obtain a spare.

7. **First Available Part is Dispensed to the First Block in the Queue**: The pool will dispense a requested part if it has one in stock or when it becomes available, regardless of what action (i.e., as needed restock or emergency restock) that request may have initiated.

   a) For example, if Block A requests a part from a pool and that triggers an emergency restock action, but a part arrives before the emergency restock through another action (e.g., scheduled restock), then the pool will dispense the newly arrived part to Block A (if Block A is next in the queue to receive a part).

8. **Blocks that Trigger an Action Get Charged with the Action**: A block that triggers an emergency restock is charged for the additional cost to obtain the emergency part, even if it does not use an emergency part (i.e., even if another part becomes available first).

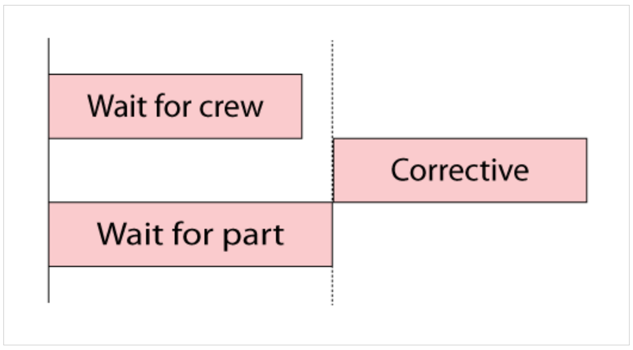
9. **Triggered Action Cannot be Canceled.** If a block triggers a restock action but then receives a part from another source, the action that the block triggered is not canceled.

   a) For example, if Block A initiates an emergency restock action but was then able to use a part that became available through other actions, the emergency request is not canceled and an emergency spare part will be added to the pool's stock level.

   b) Another way to explain this is by looking at the part acquisition logistic times as transit times. Because an ordered part is en-route to you after you order it, you will receive it regardless of whether the conditions have changed and you no longer need it.

Simultaneous Dispatch of Crews and Parts Logic

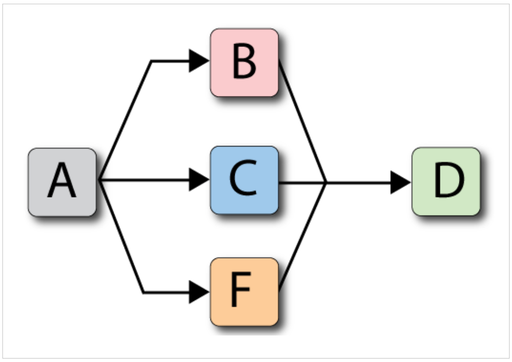
Some special rules apply when a block has both logistic delays in acquiring parts from a pool and when waiting for crews. BlockSim dispatches requests for crews and spare parts simultaneously. The repair action does not start until both crew and part arrive, as shown next.



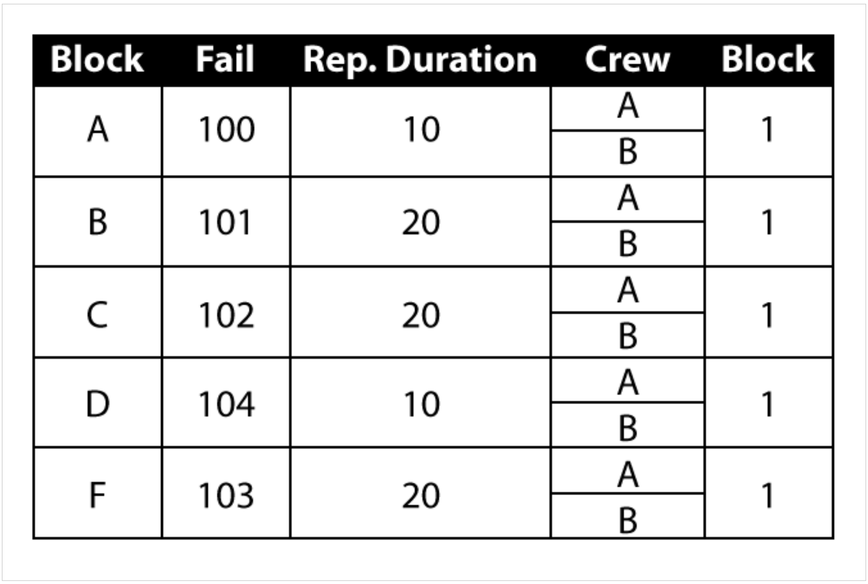
If a crew arrives and it has to wait for a part, then this time (and cost) is added to the crew usage time.

Example Using Both Crews and Pools

Consider the following example, using both crews and pools.



where:



Block	Fail	Rep. Duration	Crew	Block
A	100	10	A	1
			B	
B	101	20	A	1
			B	
C	102	20	A	1
			B	
D	104	10	A	1
			B	
F	103	20	A	1
			B	

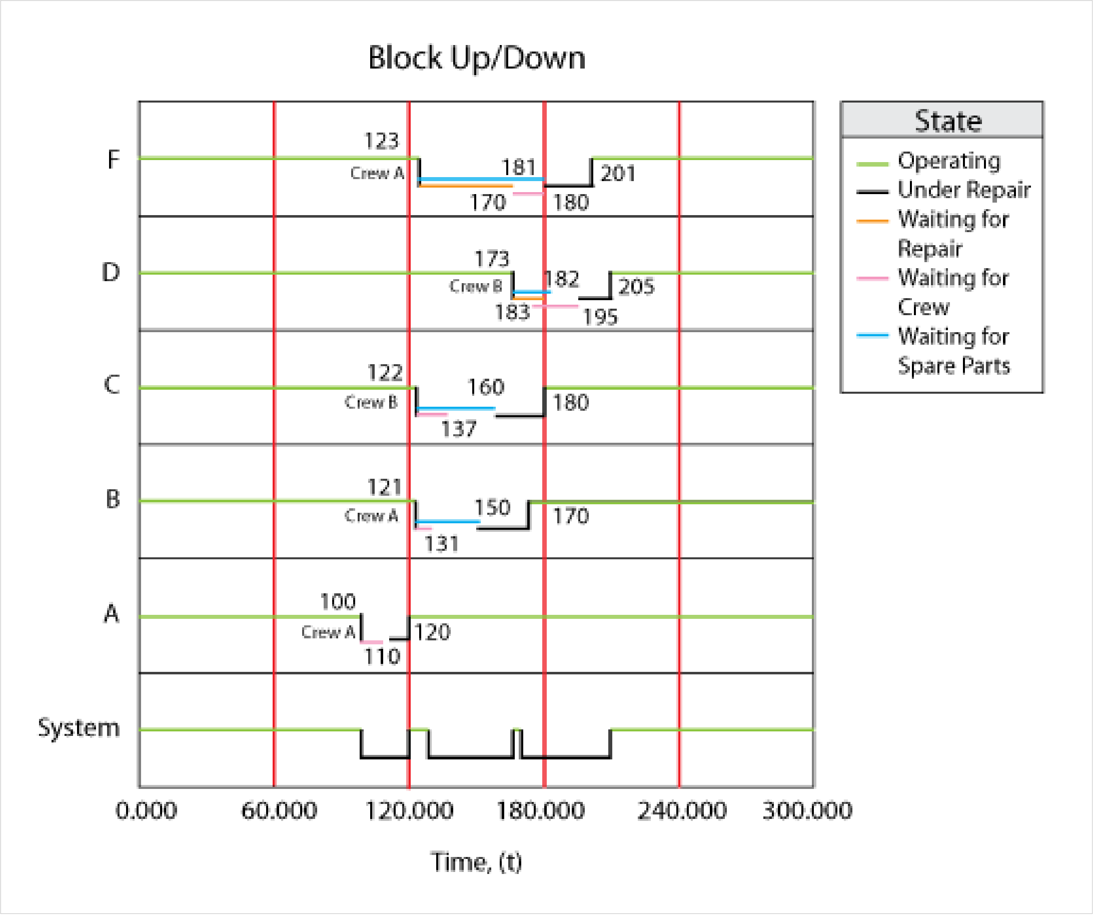
And the crews are:

Crew	Num. Sim. Tasks	Delay	Cost PI	Cost PUT
A	1	10	10	1
B	1	15	20	2

While the spare pool is:

Pool	Init. Stock	Restock	On Condition	On Condition Delay
1	1	1 every 150	When 0 Stock 1	60

The behavior of this system from 0 to 300 is shown graphically in the figure below.



The discrete system events during that time are as follows:

1. Component A fails at 100 and Crew A is engaged.

   a) At 110, Crew A arrives and completes the repair by 120.

   b) This repair uses the only spare part in inventory and triggers an on-condition restock. A part is ordered and is scheduled to arrive at 160.

   c) A scheduled restock part is also set to arrive at 150.

   d) Pool [on-hand = 0, pending: 150, 160].

2. Component B fails at 121. Crew A is available and it is engaged.

   a) Crew A arrives by 131 but no part is available.

b) The failure finds the pool with no parts, triggering the on-condition restock. A part was ordered and is scheduled to arrive at 181.

c) Pool [on-hand = 0, pending: 150, 160, 181].

d) At 150, the first part arrives and is used by Component B.

e) Repair on Component $B$is completed 20 time units later, at 170.

f) Pool [on-hand=0, pending: 160, 181].

3. Component Cfails at 122. Crew $A$is already engaged by Component B, thus Crew $B$is engaged.

a) Crew Barrives at 137 but no part is available.

b) The failure finds the pool with no parts, triggering the on-condition restock. A part is ordered and is scheduled to arrive at 182.

c) Pool [on-hand = 0, pending: 160, 181,182].

d) At 160, the part arrives and Component $C$is repaired by 180.

e) Pool [on-hand = 0, pending: 181,182].

4. Component Ffails at 123. No crews are available until 170 when Crew Abecomes available.

a) Crew Aarrives by 180 and has to wait for a part.

b) The failure found the pool with no parts, triggering the on-condition restock. A part is ordered and is scheduled to arrive at 183.

c) Pool [on-hand = 0, pending: 181,182, 183].

d) At 181, a part is obtained.

e) By 201, the repair is completed.

f) Pool [on-hand = 0, pending: 182, 183]

5. Component Dfails at 171 with no crew available.

a) Crew Bbecomes available at 180 and arrives by 195.

b) The failure finds the pool with no parts, triggering the on-condition restock. A part is ordered and is scheduled to arrive at 231.

c) The next part becomes available at 182 and the repair is completed by 205.

d) Pool [on-hand = 0, pending: 183, 231]

6. End time is at 300. The last scheduled part arrives at the pool at 300.

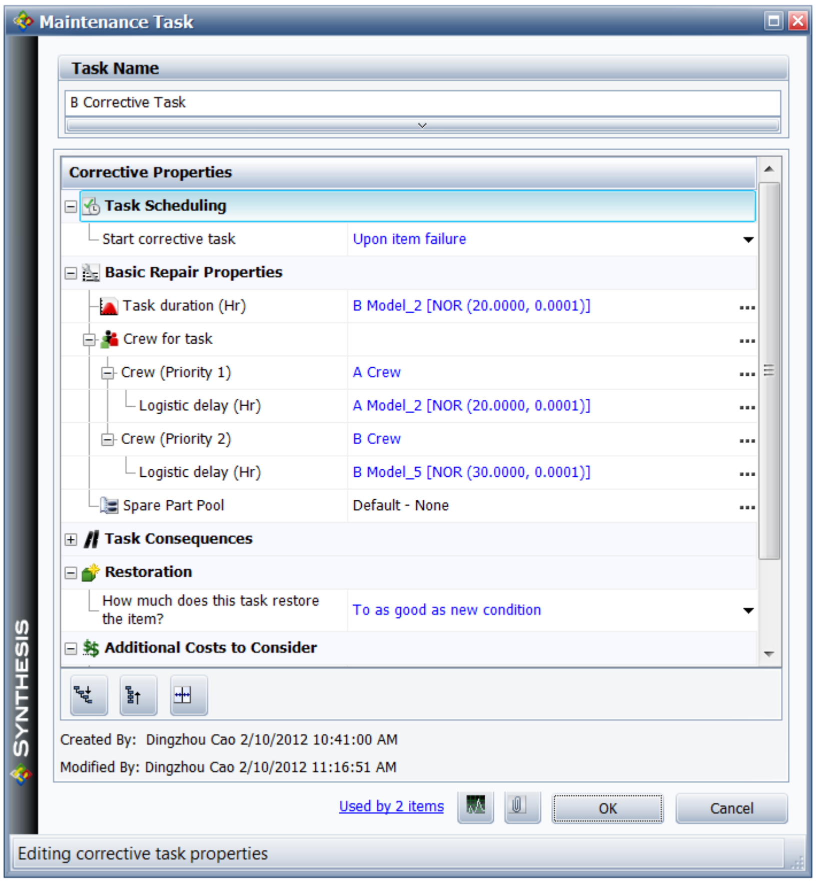
Using Maintenance Tasks

One of the most important benefits of simulation is the ability to define how and when actions are performed. In our case, the actions of interest are part repairs/replacements. This is accomplished in BlockSim through the use of maintenance tasks. Specifically, four different types of tasks can be defined for maintenance actions: corrective maintenance, preventive maintenance, on condition maintenance and inspection.

Corrective Maintenance Tasks

A corrective maintenance task defines when a corrective maintenance (CM) action is performed. The figure below shows a corrective maintenance task assigned to a block in BlockSim. Corrective actions will be performed either immediately upon failure of the item or upon finding that the item has failed (for hidden failures that are not detected until an inspection). BlockSim allows the selection of either category.

- **Upon item failure**: The CM action is initiated immediately upon failure. If the user doesn't specify the choice for a CM, then this is the default option. All prior examples were based on the instruction to perform a CM upon failure.
- **When found failed during an Inspection**: The CM action will only be initiated after an inspection is done on the failed component. How and when the inspections are performed is defined by the block's inspection properties. This has the effect of defining a dependency between the corrective maintenance task and the inspection task.



Maintenance Task

Task Name: B Corrective Task

Corrective Properties

Task Scheduling	
Start corrective task	Upon item failure
Basic Repair Properties	
Task duration (Hr)	B Model_2 [NOR (20.0000, 0.0001)]
Crew for task	
Crew (Priority 1)	A Crew
Logistic delay (Hr)	A Model_2 [NOR (20.0000, 0.0001)]
Crew (Priority 2)	B Crew
Logistic delay (Hr)	B Model_5 [NOR (30.0000, 0.0001)]
Spare Part Pool	Default - None
Task Consequences	
Restoration	
How much does this task restore the item?	To as good as new condition
Additional Costs to Consider	

Created By: Dingzhou Cao 2/10/2012 10:41:00 AM
Modified By: Dingzhou Cao 2/10/2012 11:16:51 AM

Editing corrective task properties

See It In action...

More application examples are available! See also:

CM Triggered by Subsystem Down

Scheduled Tasks

Scheduled tasks can be performed on a known schedule, which can be based on any of the following:

- A time interval, either fixed or dynamic, based on the item's age (item clock) or on calendar time (system clock). See Item and System Ages.
- The occurrence of certain events, including:
  - The system goes down.
  - Certain events happen in a maintenance group. The events and groups are user-specified, and the item that the task is assigned to does not need to be part of the selected maintenance group(s).

The types of scheduled tasks include:

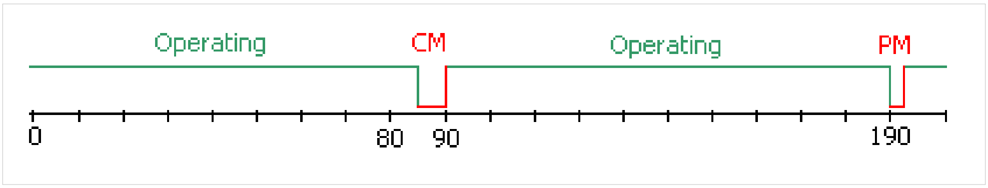
- Inspection tasks
- Preventive maintenance tasks

- On condition tasks

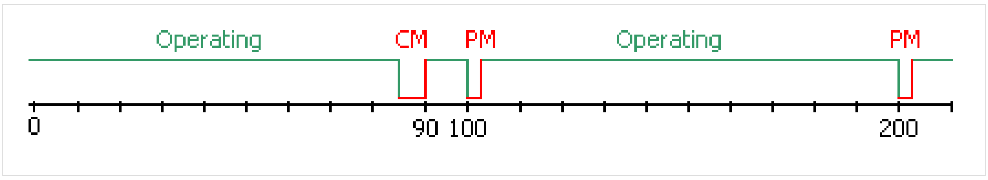
Item and System Ages

It is important to keep in mind that the system and each component of the system maintain separate clocks within the simulation. When setting intervals to perform a scheduled task, the intervals can be based on either type of clock. Specifically:

- Item age refers to the accumulated age of the block, which gets adjusted each time the block is repaired (i.e., restored). If the block is repaired at least once during the simulation, this will be different from the elapsed simulation time. For example, if the restoration factor is 1 (i.e., "as good as new") and the assigned interval is 100 days based on item age, then the task will be scheduled to be performed for the first time at 100 days of elapsed simulation time. However, if the block fails at 85 days and it takes 5 days to complete the repair, then the block will be fully restored at 90 days and its accumulated age will be reset to 0 at that point. Therefore, if another failure does not occur in the meantime, the task will be performed for the first time 100 days later at 190 days of elapsed simulation time.



- Calendar time refers to the elapsed simulation time. If the assigned interval is 100 days based on calendar time, then the task will be performed for the first time at 100 days of elapsed simulation time, for the second time at 200 days of elapsed simulation time and so on, regardless of whether the block fails and gets repaired correctively between those times.

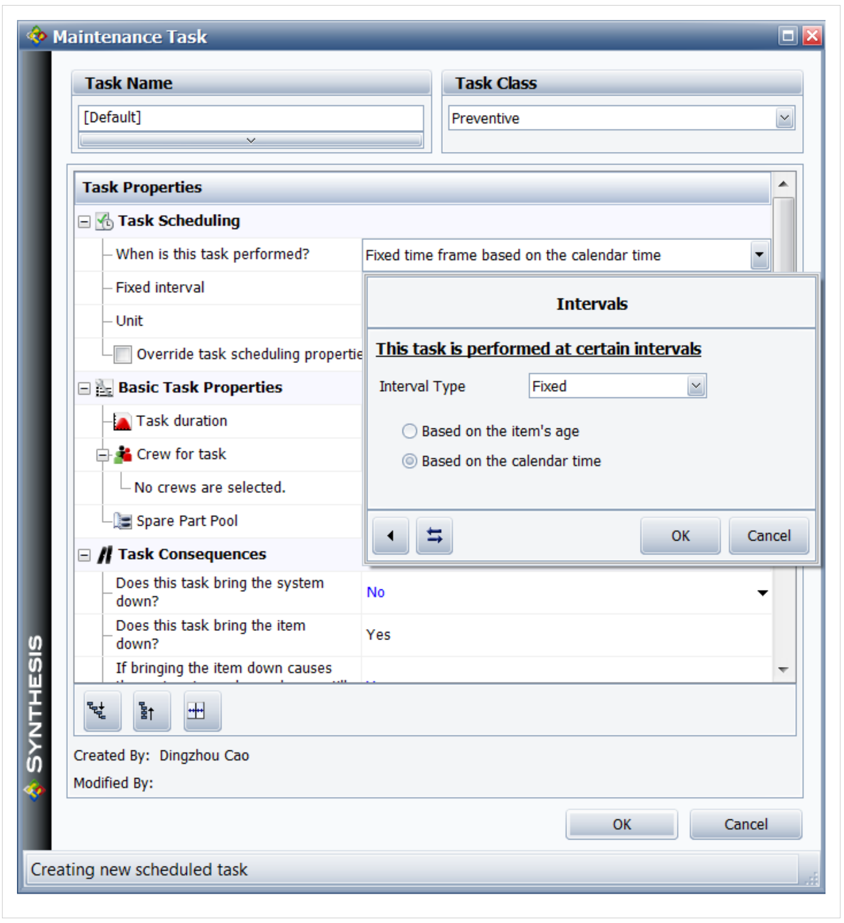


Inspection Tasks

Like all scheduled tasks, inspections can be performed based on a time interval or upon certain events. Inspections can be specified to bring the item or system down or not.

Preventive Maintenance Tasks

The figure below shows the options available in a preventive maintenance (PM) task within BlockSim. PMs can be performed based on a time interval or upon certain events. Because PM tasks always bring the item down, one can also specify whether preventive maintenance will be performed if the task brings the system down.


Maintenance Task

Task Name: [Default]

Task Class: Preventive

Task Properties

Task Scheduling
- When is this task performed? Fixed time frame based on the calendar time
- Fixed interval
- Unit
- Override task scheduling propertie

Basic Task Properties
- Task duration
- Crew for task
  - No crews are selected.
- Spare Part Pool

Task Consequences
- Does this task bring the system down? No
- Does this task bring the item down? Yes
- If bringing the item down causes

Intervals

This task is performed at certain intervals

Interval Type: Fixed

( ) Based on the item's age
(●) Based on the calendar time

Created By: Dingzhou Cao

Modified By:

Creating new scheduled task

On Condition Tasks

On condition maintenance relies on the capability to detect failures before they happen so that preventive maintenance can be initiated. If, during an inspection, maintenance personnel can find evidence that the equipment is approaching the end of its life, then it may be possible to delay the failure, prevent it from happening or replace the equipment at the earliest convenience rather then allowing the failure to occur and possibly cause severe consequences. In BlockSim, on condition tasks consist of an inspection task that triggers a preventive task when an impending failure is detected during inspection.

Failure Detection

Inspection tasks can be used to check for indications of an approaching failure. BlockSim models such indications of when an approaching failure will become detectable upon inspection using Failure Detection Threshold and P-F Interval. Failure detection threshold allows the user to enter a number between 0 and 1 indicating the percentage of an item's life that must elapse before an approaching failure can be detected. For instance, if the failure detection threshold value is set as 0.8 then this means that the failure of a component can be detected only during the last 20% of its life. If an inspection occurs during this time, an approaching failure is detected and the inspection triggers a preventive maintenance task to take the necessary precautions to delay the failure by either repairing or replacing the component.

The P-F interval allows the user to enter the amount of time before the failure of a component when the approaching failure can be detected by an inspection. The P-F interval represents the warning period that spans from P(when a potential failure can be detected) to F(when the failure occurs). If a P-F interval is set as 200 hours, then the approaching failure of the component can only be detected at 200 hours before the failure of the component. Thus, if a component has a fixed life of 1,000 hours and the P-F interval is set to 200 hours, then if an inspection occurs at or beyond 800 hours, then the approaching failure of the component that is to occur at 1,000 hours is detected by this inspection and a preventive maintenance task is triggered to take action against this failure.

Rules for On Condition Tasks

- An inspection that finds a block at or beyond the failure detection threshold or within the range of the P-F interval will trigger the associated preventive task as long as preventive maintenance can be performed on that block.
- If a non-downing inspection triggers a preventive maintenance action because the failure detection threshold or P-F interval range was reached, no other maintenance task will be performed between the inspection and the triggered preventive task; tasks that would otherwise have happened at that time due to system age, system down or group maintenance will be ignored.
- A preventive task that would have been triggered by a non-downing inspection will not happen if the block fails during the inspection, as corrective maintenance will take place instead.
- If a failure will occur within the failure detection threshold or P-F interval set for the inspection, but the preventive task is only supposed to be performed when the system is down, the simulation waits until the requirements of the preventive task are met to perform the preventive maintenance.
- If the on condition inspection triggers the preventive maintenance part of the task, the simulation assumes that the maintenance crew will forego any routine servicing associated with the inspection part of the task. In other words, the restoration will come from the preventive maintenance, so any restoration factor defined for the inspection will be ignored in these circumstances.

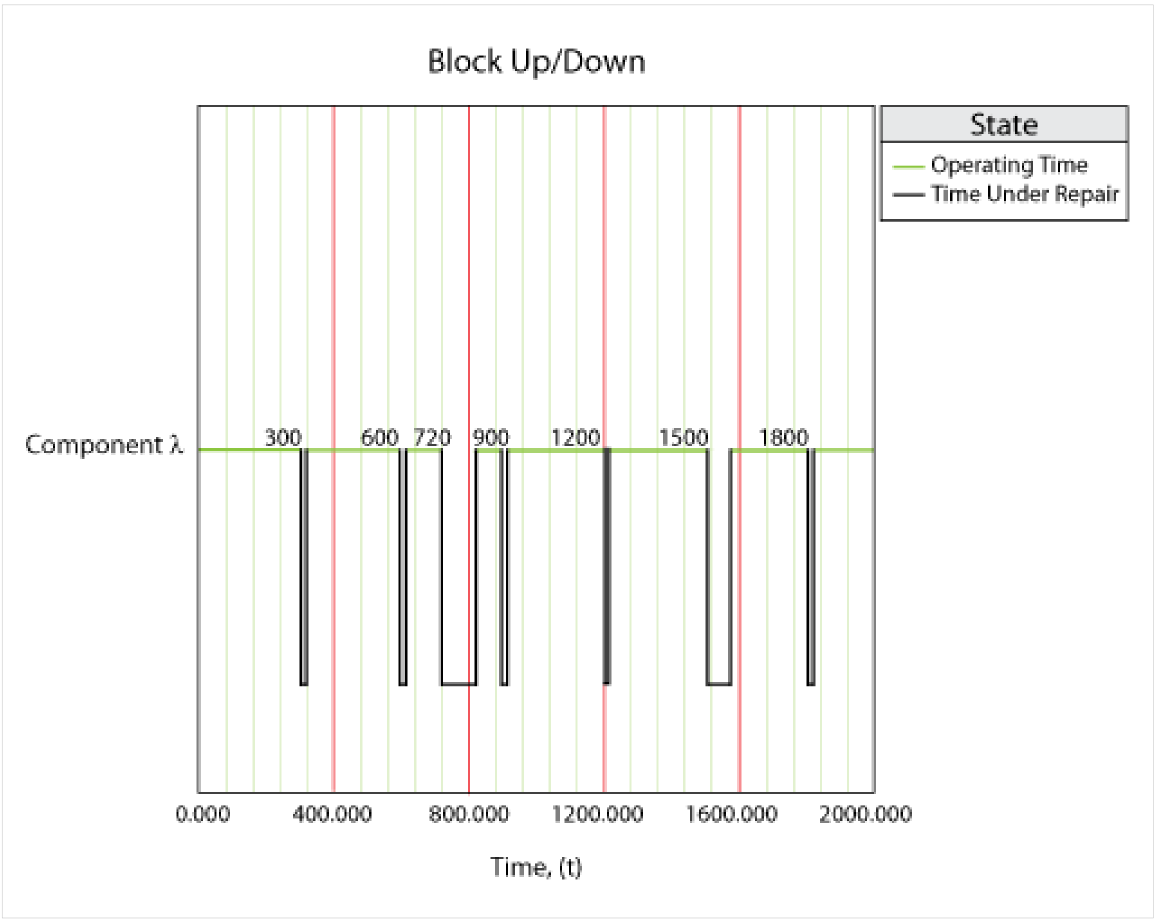
Example Using P-F Interval

To illustrate the use of the P-F interval in BlockSim, consider a component Athat fails every 700 tu. The corrective maintenance on this equipment takes 100 $tu$to complete, while the preventive maintenance takes 50 $tu$to complete. Both the corrective and preventive maintenance actions have a type II restoration factor of 1. Inspection tasks of 10 tuduration are performed on the component every 300 tu. There is no restoration of the component during the inspections. The P-F interval for this component is 100 tu.

The component behavior from 0 to 2000 $tu$is shown in the figure below and described next.

1. At 300 $tu$the first scheduled inspection of 10 tuduration occurs. At this time the age of the component is 300 tu. This inspection does not lie in the P-F interval of 100 tu(which begins at the age of 600 $tu$and ends at the age of 700 tu). Thus, no approaching failure is detected during this inspection.
2. At 600 $tu$the second scheduled inspection of 10 tuduration occurs. At this time the age of the component is 590 tu(no age is accumulated during the first inspection from 300 tu to 310 tuas the component does not operate during this inspection). Again this inspection does not lie in the P-F interval. Thus, no approaching failure is detected during this inspection.
3. At 720 $tu$the component fails after having accumulated an age of 700 tu. A corrective maintenance task of 100 tuduration occurs to restore the component to as-good-as-new condition.
4. At 900 $tu$the third scheduled inspection occurs. At this time the age of the component is 80 tu. This inspection does not lie in the P-F interval (from age 600 $tu$to 700 tu). Thus, no approaching failure is detected during this inspection.
5. At 1200 $tu$the fourth scheduled inspection occurs. At this time the age of the component is 370 tu. Again, this inspection does not lie in the P-F interval and no approaching failure is detected.
6. At 1500 $tu$the fifth scheduled inspection occurs. At this time the age of the component is 660 tu, which lies in the P-F interval. As a result, an approaching failure is detected and the inspection triggers a preventive maintenance task. A preventive maintenance task of 50 tuduration occurs at 1510 $tu$to restore the component to as-good-as-new condition.

7. At 1800 $tu$the sixth scheduled inspection occurs. At this time the age of the component is 240 tu. This inspection does not lie in the P-F interval (from age 600 tu to 700 tu) and no approaching failure is detected.



Rules for PMs and Inspections

All the options available in the Maintenance task window were designed to maximize the modeling flexibility within BlockSim. However, maximizing the modeling flexibility introduces issues that you need to be aware of and requires you to carefully select options in order to assure that the selections do not contradict one another. One obvious case would be to define a PM action on a component in series (which will always bring the system down) and then assign a PM policy to the block that has the Do not perform maintenance if the action brings the system down option set. With these settings, no PMs will ever be performed on the component during the BlockSim simulation. The following sections summarize some issues and special cases to consider when defining maintenance properties in BlockSim.

1. Inspections do not consume spare parts. However, an inspection can have a renewal effect on the component if the restoration factor is set to a number other than the default of 0.
2. On the inspection tab, if Inspection brings system down is selected, this also implies that the inspection brings the item down.
3. If a PM or an inspection are scheduled based on the item's age, then they will occur exactly when the item reaches that age. However, it is important to note that failed items do not age. Thus, if an item fails before it reaches that age, the action will not be performed. This means that if the item fails before the scheduled inspection (based on item age) and the CM is set to be performed upon inspection, the CM will never take place. The reason that this option is allowed in BlockSim is for the flexibility of specifying renewing inspections.

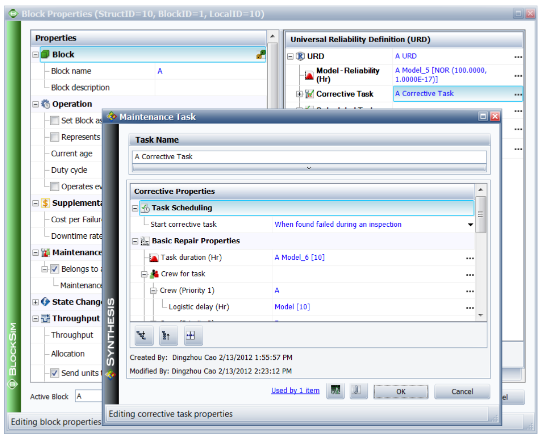
4. Downtime due to a failure discovered during a non-downing inspection is included when computing results "w/o PM, OC & Inspections."
5. If a PM upon item age is scheduled and is not performed because it brings the system down (based on the option in the PM task) the PM will not happen unless the item reaches that age again (after restoration by CM, inspection or another type of PM).
6. If the CM task is upon inspection and a failed component is scheduled for PM prior to the inspection, the PM action will restore the component and the CM will not take place.
7. In the case of simultaneous events, only one event is executed (except the case in maintenance phase, in maintenance phase, all simultaneous events in maintenance phase are executed in a order). The following precedence order is used: 1). Tasks based on intervals or upon start of a maintenance phase; 2). Tasks based on events in a maintenance group, where the triggering event applies to a block; 3). Tasks based on system down; 4). Tasked on events in a maintenance group, where the triggering event applies to a subdiagram. Within these categories, order is determined according to the priorities specified in the URD (i.e., the higher the task in on the list, the higher the priority).
8. The PM option of Do not perform if it brings the system down is only considered at the time that the PM needs to be initiated. If the system is down at that time, due to another item, then the PM will be performed regardless of any future consequences to the system up state. In other words, when the other item is fixed, it is possible that the system will remain down due to this PM action. In this case, the PM time difference is added to the system PM downtime.
9. Downing events cannot overlap. If a component is down due to a PM and another PM is suggested based on another trigger, the second call is ignored.
10. A non-downing inspection with a restoration factor restores the block based on the age of the block at the beginning of the inspection (i.e., duration is not restored).
11. Non-downing events can overlap with downing events. If in a non-downing inspection and a downing event happen concurrently, the non-downing event will be managed in parallel with the downing event.
12. If a failure or PM occurs during a non-downing inspection and the CM or PM has a restoration factor and the inspection action has a restoration factor, then both restoration factors are used (compounded).
13. A PM or inspection on system down is triggered only if the system was up at the time that the event brought the system down.
14. A non-downing inspection with restoration factor of 0 does not affect the block.

Example

To illustrate the use of maintenance policies in BlockSim we will use the same example from Example Using Both Crews and Pools with the following modifications (The figures below also show these settings):

Blocks A and D:

1. Belong to the same group (Group 1).
2. Corrective maintenance actions are upon inspection (not upon failure) and the inspections are performed every 30 hours, based on system time. Inspections have a duration of 1 hour. Furthermore, unlimited free crews are available to perform the inspections.
3. Whenever either item get CM, the other one gets a PM.
4. The PM has a fixed duration of 10 hours.
5. The same crews are used for both corrective and preventive maintenance actions.



Block name: A
Block description:

Universal Reliability Definition (URD)

URD	A URD
Model - Reliability (Hr)	A Model_5 [NOR (100.0000, 1.0000E-17)]
Corrective Task	A Corrective Task

Task Name: A Corrective Task

Corrective Properties

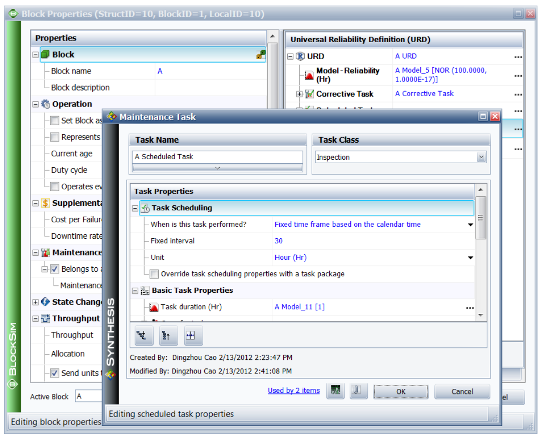
Task Scheduling	
Start corrective task	When found failed during an inspection
Basic Repair Properties	
Task duration (Hr)	A Model_6 [10]
Crew for task	
Crew (Priority 1)	A
Logistic delay (Hr)	Model [10]

Created By: Dingzhou Cao 2/13/2012 1:55:57 PM
Modified By: Dingzhou Cao 2/13/2012 2:23:12 PM

Active Block: A

Editing block properties

Editing corrective task properties



Block name: A
Block description:

Universal Reliability Definition (URD)

URD	A URD
Model - Reliability (Hr)	A Model_5 [NOR (100.0000, 1.0000E-17)]
Corrective Task	A Corrective Task

Task Name: A Scheduled Task
Task Class: Inspection

Task Properties

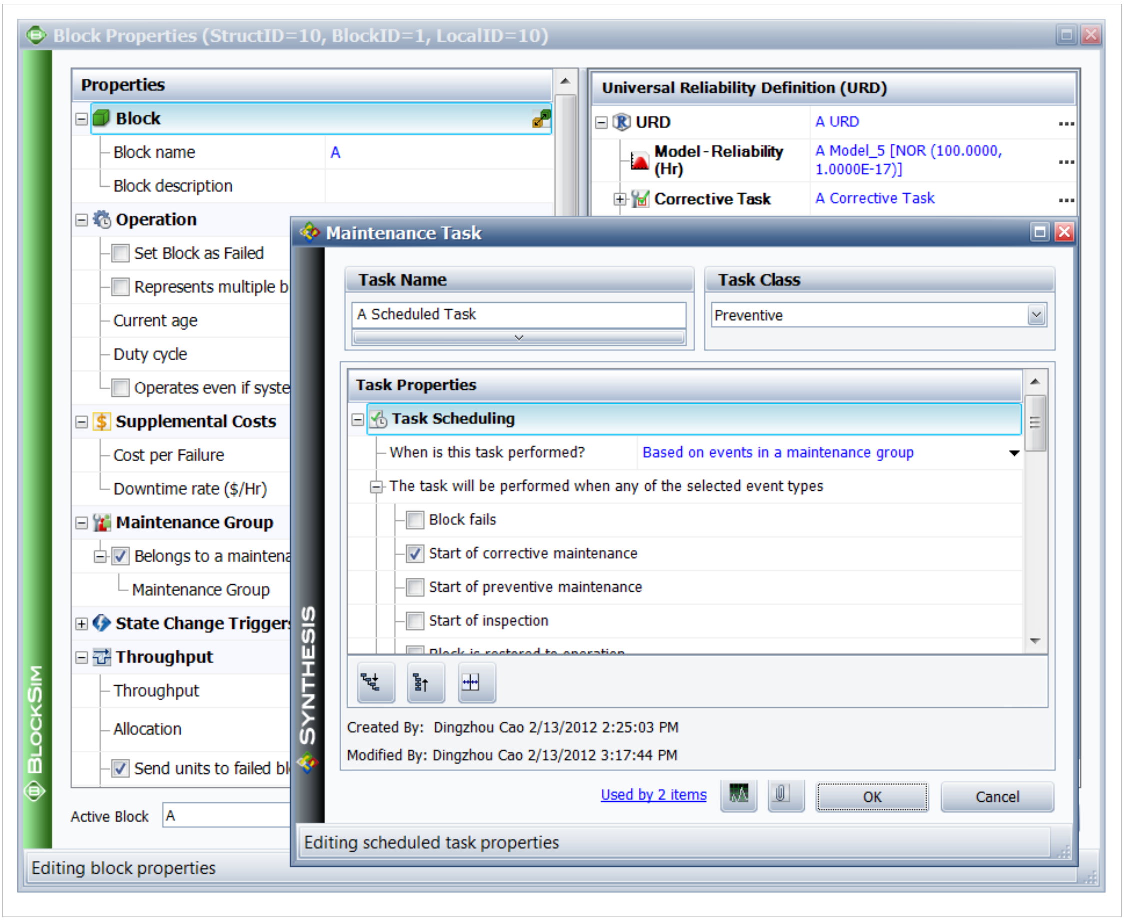
Task Scheduling	
When is this task performed?	Fixed time frame based on the calendar time
Fixed interval	30
Unit	Hour (Hr)
☐ Override task scheduling properties with a task package	
Basic Task Properties	
Task duration (Hr)	A Model_11 [1]

Created By: Dingzhou Cao 2/13/2012 2:23:47 PM
Modified By: Dingzhou Cao 2/13/2012 2:41:08 PM

Active Block: A

Editing block properties

Editing scheduled task properties


Block Properties (StructID=10, BlockID=1, LocalID=10)

Properties

Block
- Block name: A
- Block description

Operation
- Set Block as Failed
- Represents multiple b
- Current age
- Duty cycle
- Operates even if syste

Supplemental Costs
- Cost per Failure
- Downtime rate ($/Hr)

Maintenance Group
- ✓ Belongs to a maintena
- Maintenance Group

State Change Trigger

Throughput
- Throughput
- Allocation
- ✓ Send units to failed bl

Active Block: A

Editing block properties

Universal Reliability Definition (URD)

URD	A URD	...
Model - Reliability (Hr)	A Model_5 [NOR (100.0000, 1.0000E-17)]	...
Corrective Task	A Corrective Task	...

Maintenance Task

Task Name: A Scheduled Task

Task Class: Preventive

Task Properties

Task Scheduling
- When is this task performed? Based on events in a maintenance group
- The task will be performed when any of the selected event types
  - Block fails
  - ✓ Start of corrective maintenance
  - Start of preventive maintenance
  - Start of inspection
  - Block is restored to operation

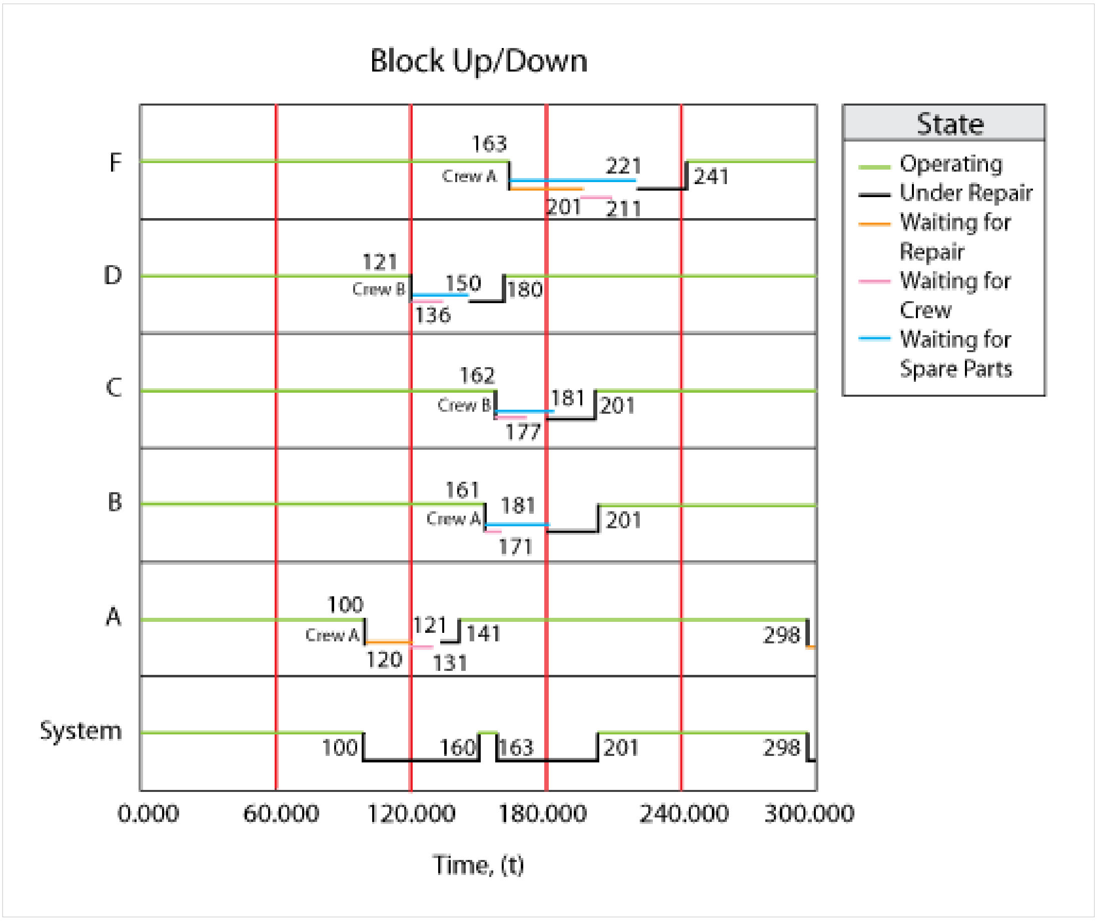
Created By: Dingzhou Cao 2/13/2012 2:25:03 PM

Modified By: Dingzhou Cao 2/13/2012 3:17:44 PM

Editing scheduled task properties

System Overview

The item and system behavior from 0 to 300 hours is shown in the figure below and described next.



1. At 100, block Agoes down and brings the system down.

   a) No maintenance action is performed since an upon inspection policy was used.

   b) The next scheduled inspection is at 120, thus Crew $A$is called to perform the maintenance by 121 (end of the inspection).

2. Crew Aarrives and initiates the repair on Aat 131.

   a) The only part in the pool is used and an on-condition restock is triggered.

   b) Pool [on-hand = 0, pending: 150 ^{s}, 181].

   c) Block $A$is repaired by 141.

3. At the same time (121), a PM is initiated for block Dbecause the PM task called for "PM upon the start of corrective maintenance on another group item."

   a) Crew $B$is called for block $D$and arrives at 136.

   b) No part is available until 150. An on-condition restock is triggered for 181.

   c) Pool [on-hand = 0, pending: 150 ^{s}, 181, 181].

   d) At 150, a part becomes available and the PM is completed by 160.

   e) Pool [on-hand = 0, pending: 181, 181].

4. At 161, block Bfails (corrective maintenance upon failure).

   a) Block Bgets Crew A, which arrives at 171.

   b) No part is available until 181. An on-condition restock is triggered for 221.

c) Pool [on-hand = 0, pending: 181, 181, 221].

d) A part arrives at 181.

e) The repair is completed by 201.

f) Pool [on-hand = 0, pending: 181, 221].

5. At 162, block C fails.

a) Block C gets Crew B, which arrives at 177.

b) No part is available until 181. An on-condition restock is triggered for 222.

c) Pool [on-hand = 0, pending: 181, 221, 222].

d) A part arrives at 181.

e) The repair is completed by 201.

f) Pool [on-hand = 0, pending: 221, 222].

6. At 163, block F fails and brings the system down.

a) Block F calls Crew A then B. Both are busy.

b) Crew A will be the first available so .. calls A again and waits.

c) No part is available until 221. An on-condition restock is triggered for 223.

d) Pool [on-hand = 0, pending: 221, 222, 223].

e) Crew A arrives at 211.

f) Repair begins at 221.

g) Repair is completed by 241.

h) Pool [on-hand = 0, pending: 222, 223].

7. At 298, block A goes down and brings the system down.

System Uptimes/Downtimes

1. Uptime: This is 200 hours.

a) This can be obtained by observing the following system up durations: 0 to 100, 160 to 163 and 201 to 298.

2. CM Downtime: This is 58 hours.

a) Observe that even though the system failed at 100, the CM action (on block A) was initiated at 121 and lasted until 141, thus only 20 hours of this downtime are attributed to the CM action.

b) The next CM action started at 163 when block F failed and lasted until 201 when blocks B and C were restored, thus adding another 38 hours of CM downtime.

3. Inspection Downtime: This is 1 hour.

a) The only time the system was under inspection was from 120 to 121, during the inspection of block A.

4. PM Downtime: This is 19 hours.

a) Note that the entire PM action duration on block D was from 121 to 160.

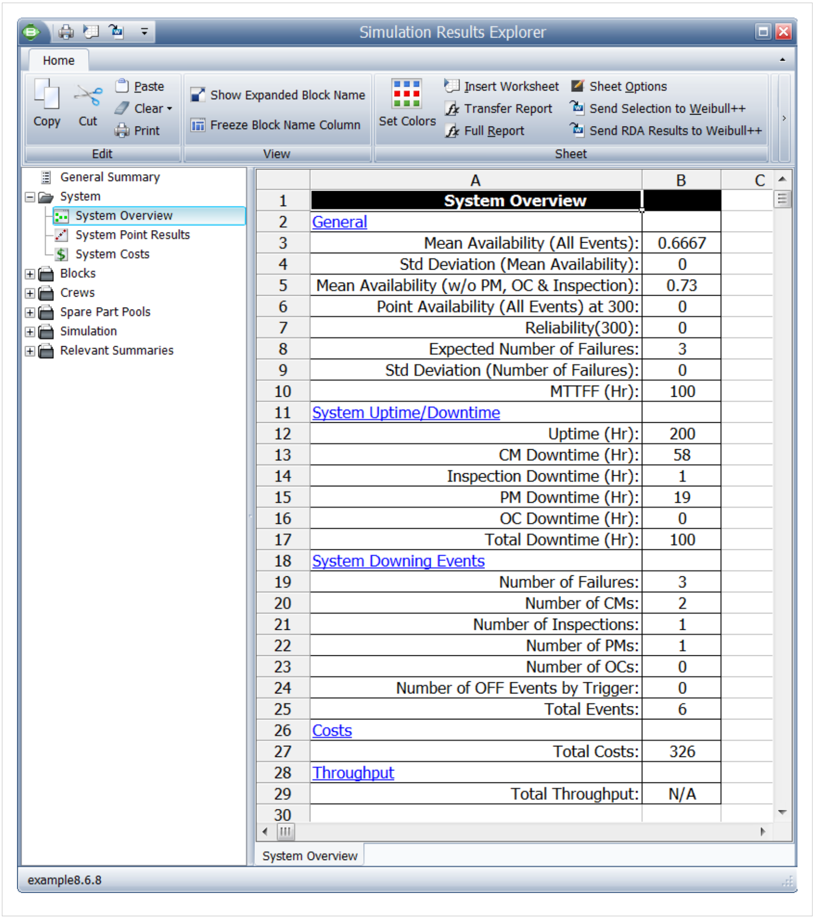
b) Until 141, and from the system perspective, the CM on block A was the cause for the downing. Once block A was restored (at 141), then the reason for the system being down became the PM on block D.

c) Thus, the PM on block D was only responsible for the downtime after block A was restored, or from 141 to 160.

5. OC Downtime: This is 0. There is not on condition task in this example.

6. Total Downtime: This is 100 hours.

   a) This includes all of the above downtimes plus the 20 hours (100 to 120) and the 2 hours (298 to 300) that the system was down due the undiscovered failure of block A.



	A	B
1	System Overview	
2	General	
3	Mean Availability (All Events):	0.6667
4	Std Deviation (Mean Availability):	0
5	Mean Availability (w/o PM, OC & Inspection):	0.73
6	Point Availability (All Events) at 300:	0
7	Reliability(300):	0
8	Expected Number of Failures:	3
9	Std Deviation (Number of Failures):	0
10	MTTFF (Hr):	100
11	System Uptime/Downtime	
12	Uptime (Hr):	200
13	CM Downtime (Hr):	58
14	Inspection Downtime (Hr):	1
15	PM Downtime (Hr):	19
16	OC Downtime (Hr):	0
17	Total Downtime (Hr):	100
18	System Downing Events	
19	Number of Failures:	3
20	Number of CMs:	2
21	Number of Inspections:	1
22	Number of PMs:	1
23	Number of OCs:	0
24	Number of OFF Events by Trigger:	0
25	Total Events:	6
26	Costs	
27	Total Costs:	326
28	Throughput	
29	Total Throughput:	N/A
30		

System Metrics

1. Mean Availability (All Events):

$$\frac{300-100}{300}=0.6667$$

2. Mean Availability (w/o PM & Inspection):

   a) This is due to the CM downtime of 58, the undiscovered downtime of 22 and the inspection downtime of 1, or:

$$\frac{300-(58+22+1)}{300}=0.7333$$

   b) It should be noted that the inspection downtime was included even though the definition was "w/o PM & Inspection." The reason for this is that the inspection did not cause the downtime in this case. Only downtimes caused by the PM or inspections are excluded.

3. Point Availability and Reliability at 300 is zero because the system was down at 300.

4. Expected Number of Failures is 3.

   a) The system failed at 100, 163 and 298.

5. The standard deviation of the number of failures is 0.

6. The MTTFF is 100 because the example is deterministic.

The System Downing Events

1. Number of Failures is 3.

   a) The first is the failure of block A, the second is the failure of block $F$and the third is the failure of block A.

2. Number of CMs is 2.

   a) The first is the CM on block $A$and the second is the CM on block F.

3. Number of Inspections is 1.

4. Number of PMs is 1.

5. Total Events are 6. These are events that the downtime can be attributed to. Specifically, the following events were observed:

   a) The failure of block Aat 100.

   b) Inspection on block Aat 120.

   c) The CM action on block A.

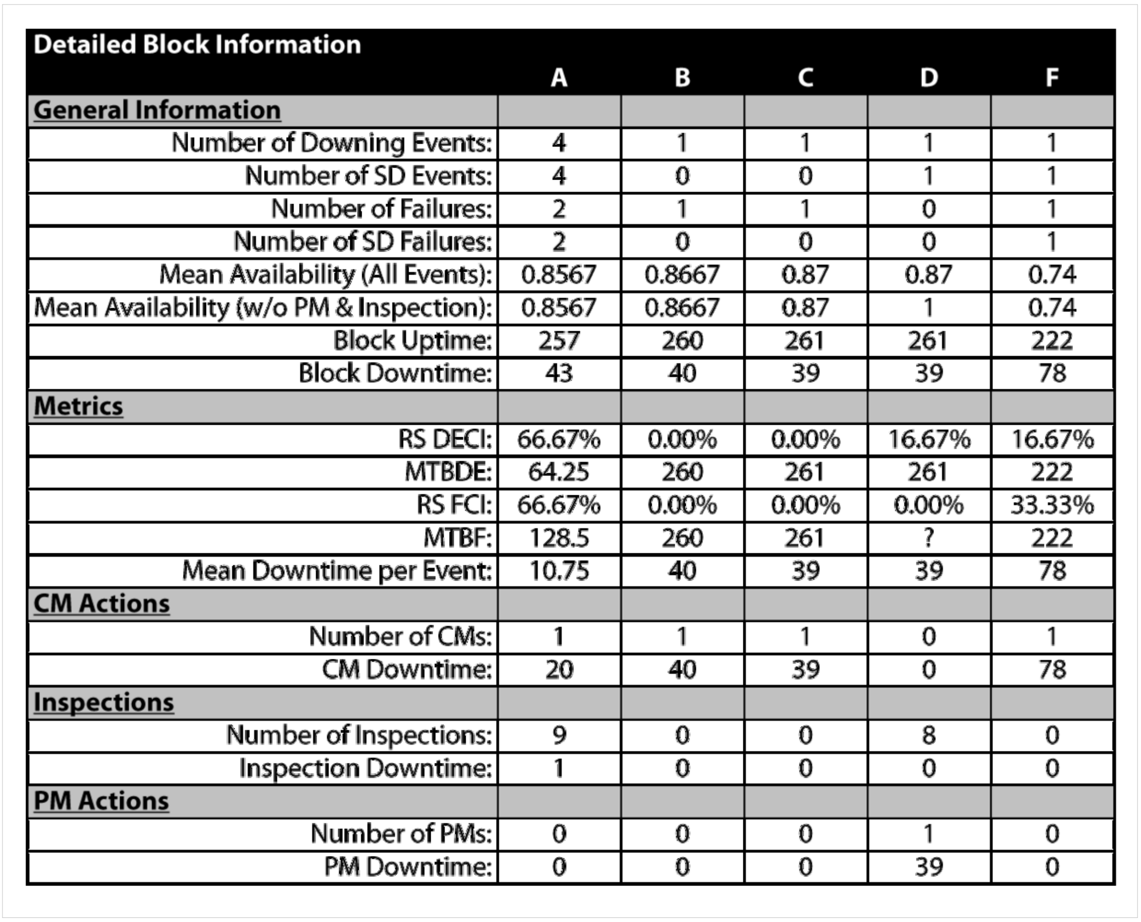
   d) The PM action on block D(after $A$was fixed).

   e) The failure of block Fat 163.

   f) The failure of block Aat 298.

Block Details

The details for blocks $A, B, C, D$and $F$are shown below.



Detailed Block Information	A	B	C	D	F
General Information					
Number of Downing Events:	4	1	1	1	1
Number of SD Events:	4	0	0	1	1
Number of Failures:	2	1	1	0	1
Number of SD Failures:	2	0	0	0	1
Mean Availability (All Events):	0.8567	0.8667	0.87	0.87	0.74
Mean Availability (w/o PM & Inspection):	0.8567	0.8667	0.87	1	0.74
Block Uptime:	257	260	261	261	222
Block Downtime:	43	40	39	39	78
Metrics					
RS DECI:	66.67%	0.00%	0.00%	16.67%	16.67%
MTBDE:	64.25	260	261	261	222
RS FCI:	66.67%	0.00%	0.00%	0.00%	33.33%
MTBF:	128.5	260	261	?	222
Mean Downtime per Event:	10.75	40	39	39	78
CM Actions					
Number of CMs:	1	1	1	0	1
CM Downtime:	20	40	39	0	78
Inspections					
Number of Inspections:	9	0	0	8	0
Inspection Downtime:	1	0	0	0	0
PM Actions					
Number of PMs:	0	0	0	1	0
PM Downtime:	0	0	0	39	0

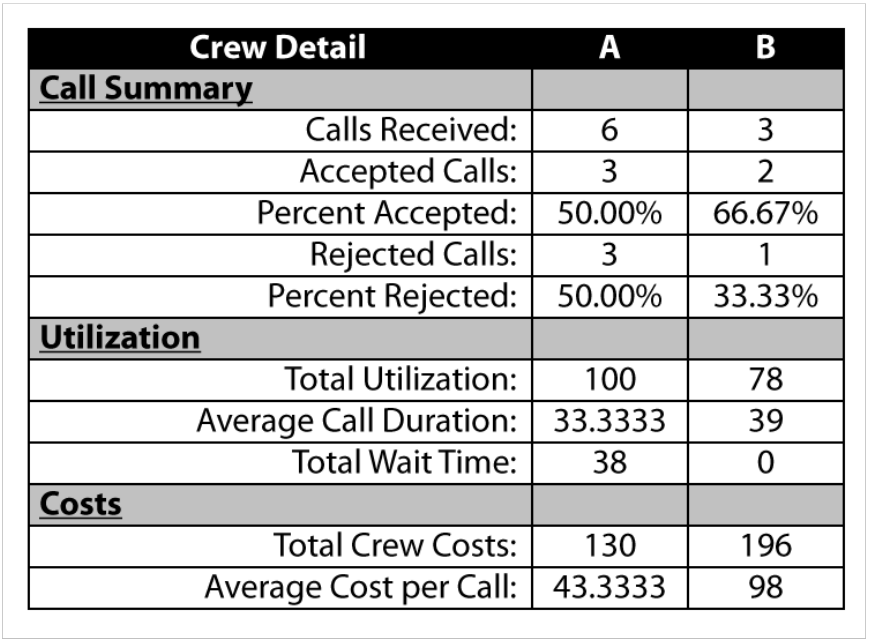
We will discuss some of these results. First note that there are four downing events on block A: initial failure, inspection and CM, plus the last failure at 298. All others have just one. Also, block Ahad a total downtime of

$41 + 2$, giving it a mean availability of 0.8567. The first time-to-failure for block Aoccurred at 100 while the second occurred after $298 - 141 = 157$hours of operation, yielding an average time between failures (MTBF) of $257/2 = 128.5$. (Note th uptime/failures.) Block Dnever failed, so its MTBF cannot be determined. Furthermore, MTBDE for each item is determined by dividing the block's uptime by the number of events. The RS FCI and RS DECI metrics are obtained by looking at the SD Failures and SD Events of the item and the number of system failures and events. Specifically, the only items that caused system failure are blocks $A$and F; Aat 100 and 298 and Fat 163. It is important to note that even though one could argue that block Falone did not cause the failure ($B$and Cwere also failed), the downing was attributed to Fbecause the system reached a failed state only when block Ffailed.

On the number of inspections, which were scheduled every 30 hours, nine occurred for block A[30, 60, 90, 120, 150, 180, 210, 240, 270] and eight for block D. Block Ddid not get inspected at 150 because block $D$was undergoing a PM action at that time.

Crew Details

The figure below shows the crew results.



Crew Detail	A	B
Call Summary		
Calls Received:	6	3
Accepted Calls:	3	2
Percent Accepted:	50.00%	66.67%
Rejected Calls:	3	1
Percent Rejected:	50.00%	33.33%
Utilization		
Total Utilization:	100	78
Average Call Duration:	33.3333	39
Total Wait Time:	38	0
Costs		
Total Crew Costs:	130	196
Average Cost per Call:	43.3333	98

Crew Areceived a total of six calls and accepted three. Specifically,

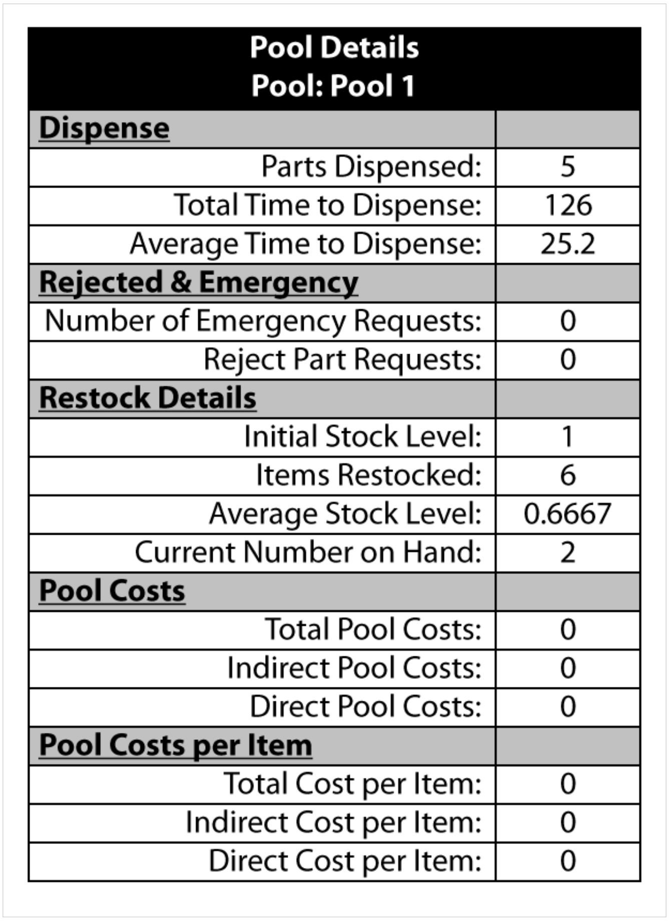
1. At 121, the crew was called by block $A$and the call was accepted.
2. At 121, block Dalso called for its PM action and was rejected. Block Dthen called crew B, which accepted the call.
3. At 161, block Bcalled crew A. Crew Aaccepted.
4. At 162, block Ccalled crew A. Crew Arejected and block Ccalled crew B, which accepted the call.
5. At 163, block Fcalled crew $A$and then crew $B$and both rejected. Block Fthen waited until a crew became available at 201 and called that crew again. This was crew A, which accepted.

The total wait time is the time that blocks had to wait for the maintenance crew. Block $F$is the only component that waited, waiting 38 hours for crew A.

Also, the costs for crew Awere 1 per unit time and 10 per incident, thus the total costs were 100 + 30. The costs for Crew Bwere 2 per unit time and 20 per incident, thus the total costs were 156 + 40.

Pool Details

The figure below shows the spare part pool results.



Pool Details Pool: Pool 1	
Dispense	
Parts Dispensed:	5
Total Time to Dispense:	126
Average Time to Dispense:	25.2
Rejected & Emergency	
Number of Emergency Requests:	0
Reject Part Requests:	0
Restock Details	
Initial Stock Level:	1
Items Restocked:	6
Average Stock Level:	0.6667
Current Number on Hand:	2
Pool Costs	
Total Pool Costs:	0
Indirect Pool Costs:	0
Direct Pool Costs:	0
Pool Costs per Item	
Total Cost per Item:	0
Indirect Cost per Item:	0
Direct Cost per Item:	0

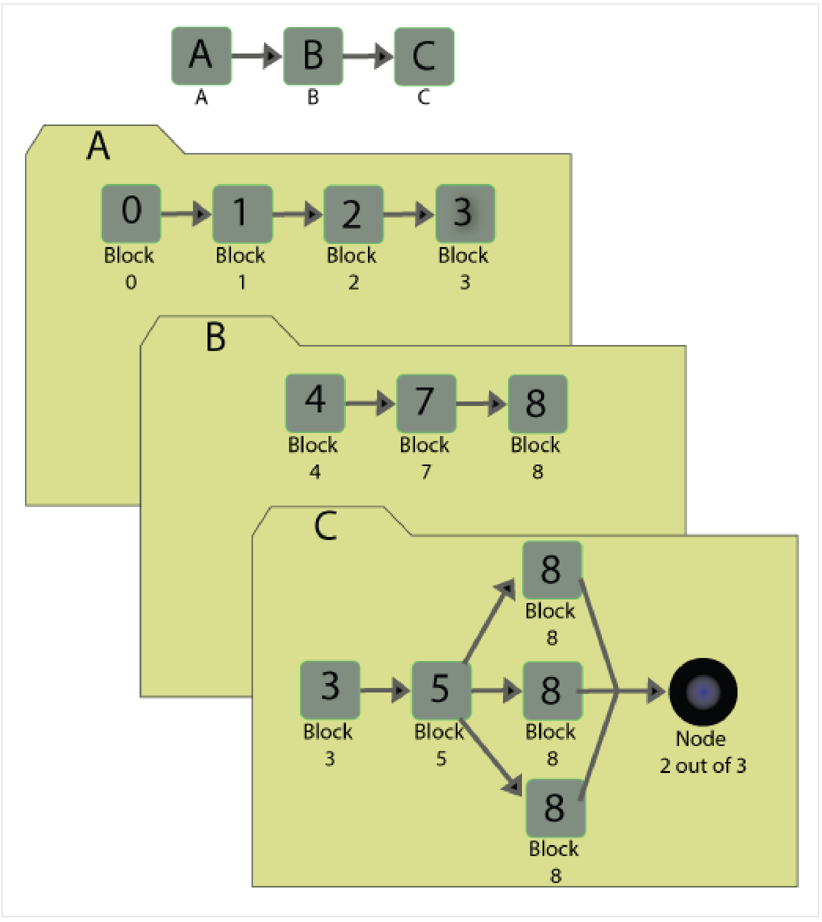
The pool started with a stock level of 1 and ended up with 2. Specifically,

1. At 121, the pool dispensed a part to block $A$and ordered another to arrive at 181.
2. At 121, it dispensed a part to block $D$and ordered another to arrive at 181.
3. At 150, a scheduled part arrived to restock the pool.
4. At 161 the pool dispensed a part to block $B$and ordered another to arrive at 221.
5. At 181, it dispensed a part to block $C$and ordered another to arrive at 222.
6. At 221, it dispensed a part to block $F$and ordered another to arrive at 223.
7. The 222 and 223 arrivals remained in stock until the end of the simulation.

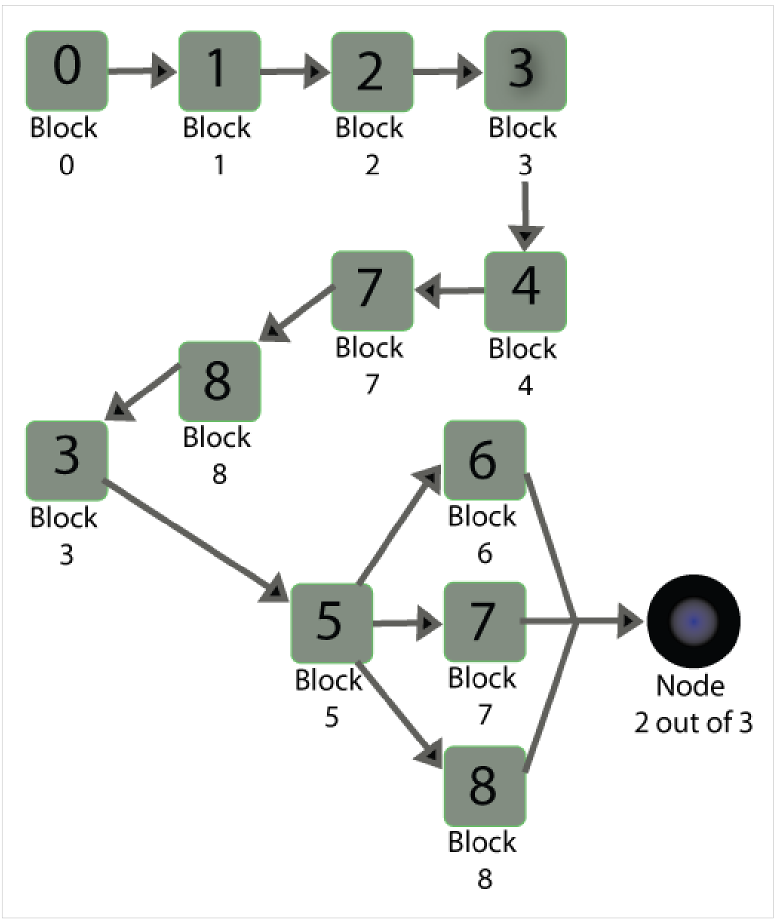
Overall, five parts were dispensed. Blocks had to wait a total of 126 hours to receive parts (B: 181-161=20, C: 181-162=19, D: 150-121=29 and F: 221-163=58).

Subdiagrams and Multi Blocks in Simulation

Any subdiagrams and multi blocks that may be present in the BlockSim RBD are expanded and/or merged into a single diagram before the system is simulated. As an example, consider the system shown in the figure below.



BlockSim will internally merge the system into a single diagram before the simulation, as shown in the figure below. This means that all the failure and repair properties of the items in the subdiagrams are also considered.

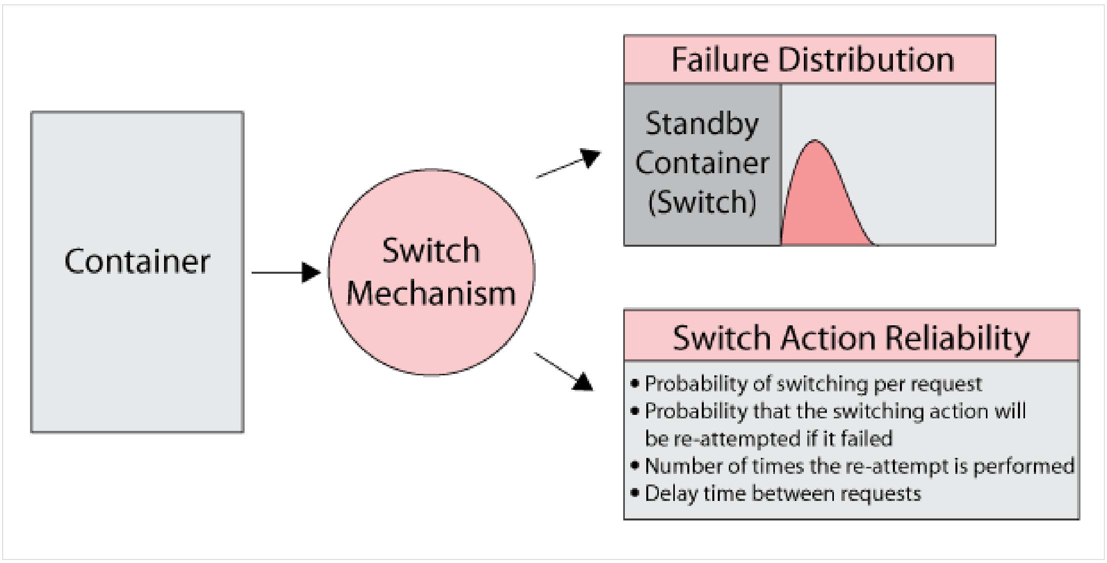


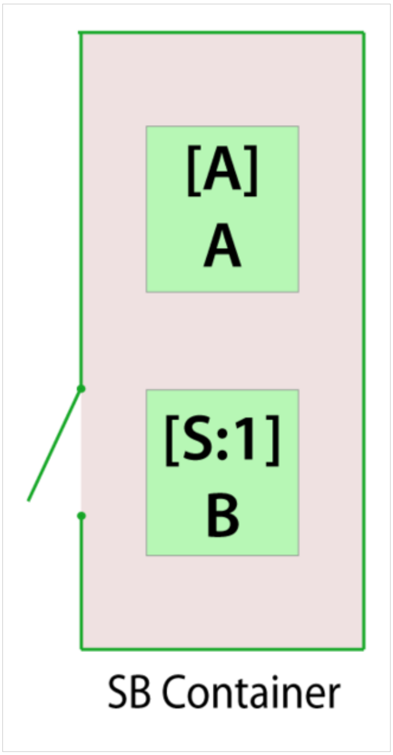
In the case of multi blocks, the blocks are also fully expanded before simulation. This means that unlike the analytical solution, the execution speed (and memory requirements) for a multi block representing ten blocks in series is identical to the representation of ten individual blocks in series.

Containers in Simulation

Standby Containers

When you simulate a diagram that contains a standby container, the container acts as the switch mechanism (as shown below) in addition to defining the standby relationships and the number of active units that are required. The container's failure and repair properties are really that of the switch itself. The switch can fail with a distribution, while waiting to switch or during the switch action. Repair properties restore the switch regardless of how the switch failed. Failure of the switch itself does not bring the container down because the switch is not really needed unless called upon to switch. The container will go down if the units within the container fail or the switch is failed when a switch action is needed. The restoration time for this is based on the repair distributions of the contained units and the switch. Furthermore, the container is down during a switch process that has a delay.


Container
Switch Mechanism
Failure Distribution
Standby Container (Switch)
Switch Action Reliability
• Probability of switching per request
• Probability that the switching action will be re-attempted if it failed
• Number of times the re-attempt is performed
• Delay time between requests



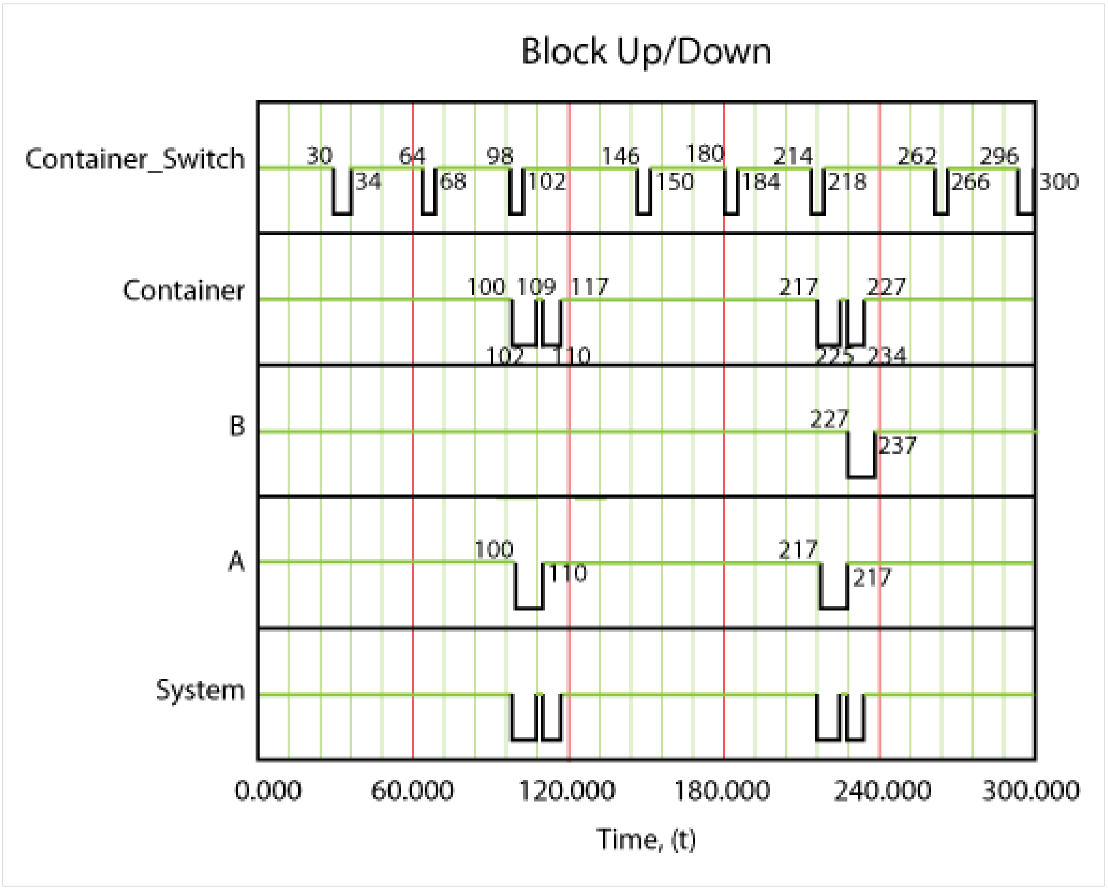
To better illustrate this, consider the following deterministic case.

1. Units A and B are contained in a standby container.
2. The standby container is the only item in the diagram, thus failure of the container is the same as failure of the system.
3. A is the active unit and B is the standby unit.
4. Unit A fails every 100 tu (active) and takes 10 tu to repair.
5. B fails every 3 tu (active) and also takes 10 tu to repair.
6. The units cannot fail while in quiescent (standby) mode.
7. Furthermore, assume that the container (acting as the switch) fails every 30 tu while waiting to switch and takes 4 tu to repair. If not failed, the container switches with 100% probability.
8. The switch action takes 7 tu to complete.
9. After repair, unit A is always reactivated.

10. The container does not operate through system failure and thus the components do not either.

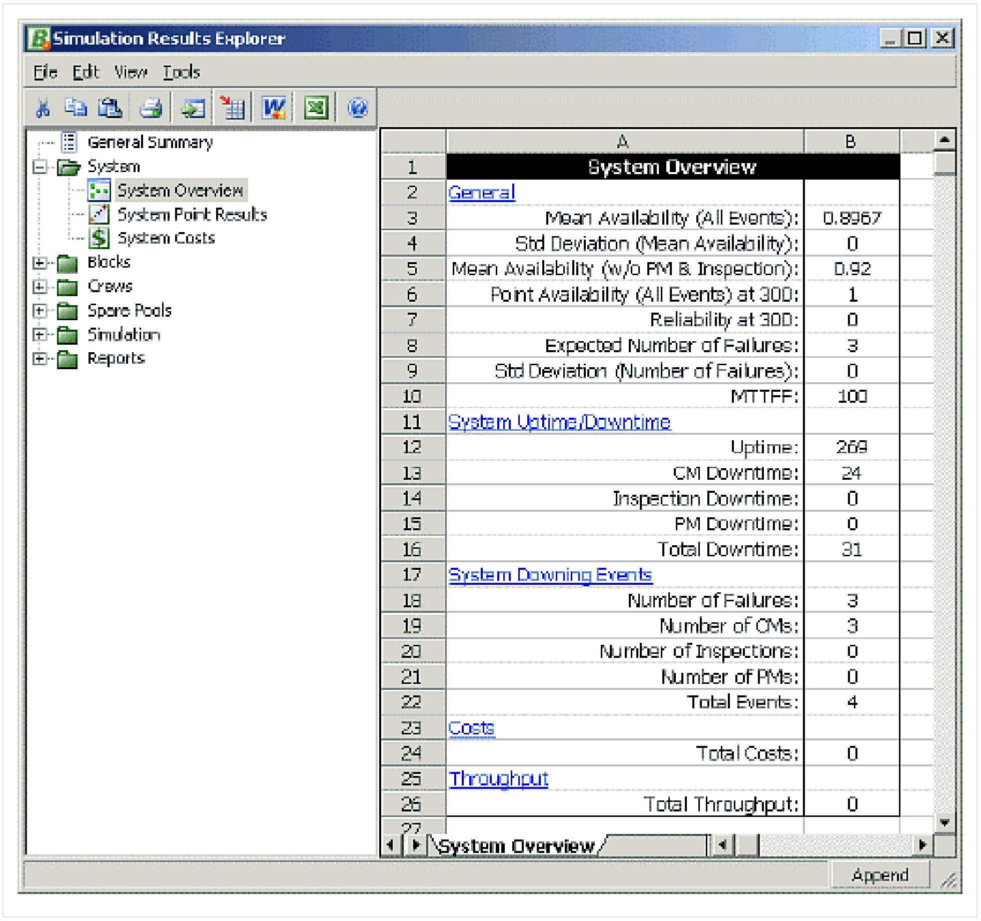
Keep in mind that we are looking at two events on the container. The container down and container switch down.

The system event log is shown in the figure below and is as follows:



1. At 30, the switch fails and gets repaired by 34. The container switch is failed and being repaired; however, the container is up during this time.
2. At 64, the switch fails and gets repaired by 68. The container is up during this time.
3. At 98, the switch fails. It will be repaired by 102.
4. At 100, unit Afails. Unit Aattempts to activate the switch to go to B; however, the switch is failed.
5. At 102, the switch is operational.
6. From 102 to 109, the switch is in the process of switching from unit $A$to unit B. The container and system are down from 100 to 109.
7. By 110, unit $A$is fixed and the system is switched back to $A$from B. The return switch action brings the container down for 7 tu, from 110 to 117. During this time, note that unit Bhas only functioned for 1 tu, 109 to 110.
8. At 146, the switch fails and gets repaired by 150. The container is up during this time.
9. At 180, the switch fails and gets repaired by 184. The container is up during this time.
10. At 214, the switch fails and gets repaired by 218.
11. At 217, unit Afails. The switch is failed at this time.
12. At 218, the switch is operational and the system is switched to unit Bwithin 7 tu. The container is down from 218 to 225.
13. At 225, unit Btakes over. After 2 $tu$of operation at 227, unit Bfails. It will be restored by 237.
14. At 227, unit $A$is repaired and the switchback action to unit $A$is initiated. By 234, the system is up.
15. At 262, the switch fails and gets repaired by 266. The container is up during this time.
16. At 296, the switch fails and gets repaired by 300. The container is up during this time.

The system results are shown in the figure below and discussed next.



	A	B
1	System Overview	
2	General	
3	Mean Availability (All Events):	0.8967
4	Std Deviation (Mean Availability):	0
5	Mean Availability (w/o PM & Inspection):	0.92
6	Point Availability (All Events) at 300:	1
7	Reliability at 300:	0
8	Expected Number of Failures:	3
9	Std Deviation (Number of Failures):	0
10	MTTFF:	100
11	System Uptime/Downtime	
12	Uptime:	269
13	CM Downtime:	24
14	Inspection Downtime:	0
15	PM Downtime:	0
16	Total Downtime:	31
17	System Downing Events	
18	Number of Failures:	3
19	Number of CMs:	3
20	Number of Inspections:	0
21	Number of PMs:	0
22	Total Events:	4
23	Costs	
24	Total Costs:	0
25	Throughput	
26	Total Throughput:	0

1. System CM Downtime is 24.

   a) CM downtime includes all downtime due to failures as well as the delay in switching from a failed active unit to a standby unit. It does not include the switchback time from the standby to the restored active unit. Thus, the times from 100 to 109, 217 to 225 and 227 to 234 are included. The time to switchback, 110 to 117, is not included.

2. System Total Downtime is 31.

   a) It includes the CM downtime and the switchback downtime.

3. Number of System Failures is 3.

   a) It includes the failures at 100, 217 and 227.

   b) This is the same as the number of CM downing events.

4. The Total Downing Events are 4.

   a) This includes the switchback downing event at 110.

5. The Mean Availability (w/o PM and Inspection) does not include the downtime due to the switchback event.

Additional Rules and Assumptions for Standby Containers

1) A container will only attempt to switch if there is an available non-failed item to switch to. If there is no such item, it will then switch if and when an item becomes available. The switch will cancel the action if it gets restored before an item becomes available.

a) As an example, consider the case of unit Afailing active while unit Bfailed in a quiescent mode. If unit Bgets restored before unit A, then the switch will be initiated. If unit $A$is restored before unit B, the switch action will not occur.

2) In cases where not all active units are required, a switch will only occur if the failed combination causes the container to fail.

a) For example, if A, $B$and $C$are in a container for which one unit is required to be operating and $A$and $B$are active with Con standby, then the failure of either Aor Bwill not cause a switching action. The container will switch to Conly if both $A$and $B$are failed.

3) If the container switch is failed and a switching action is required, the switching action will occur after the switch has been restored if it is still required (i.e., if the active unit is still failed).

4) If a switch fails during the delay time of the switching action based on the reliability distribution (quiescent failure mode), the action is still carried out unless a failure based on the switch probability/restarts occurs when attempting to switch.

5) During switching events, the change from the operating to quiescent distribution (and vice versa) occurs at the end of the delay time.

6) The option of whether components operate while the system is down is defined at component level now (This is different from BlockSim 7, in which this option of the contained items inherit from container). Two rules here:

a) If a path inside the container is down, blocks inside the container that are in that path do not continue to operate.

b) Blocks that are up do not continue to operate while the container is down.

7) A switch can have a repair distribution and maintenance properties without having a reliability distribution.

a) This is because maintenance actions are performed regardless of whether the switch failed while waiting to switch (reliability distribution) or during the actual switching process (fixed probability).

8) A switch fails during switching when the restarts are exhausted.

9) A restart is executed every time the switch fails to switch (based on its fixed probability of switching).

10) If a delay is specified, restarts happen after the delay.

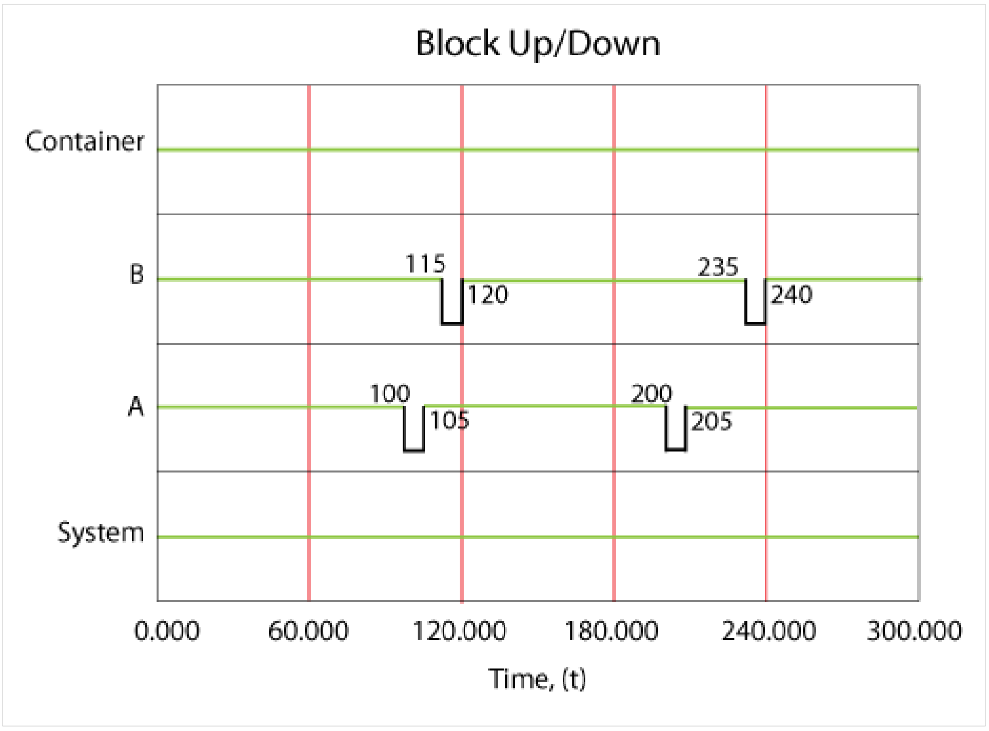
11) If a container brings the system down, the container is responsible for the system going down (not the blocks inside the container).

Load Sharing Containers

When you simulate a diagram that contains a load sharing container, the container defines the load that is shared. A load sharing container has no failure or repair distributions. The container itself is considered failed if all the blocks inside the container have failed (or $k$blocks in a k-out-of- $n$configuration).

To illustrate this, consider the following container with items $A$and Bin a load sharing redundancy.

Assume that Afails every 100 $tu$and Bevery 120 tuif both items are operating and they fail in half that time if either is operating alone (i.e., the items age twice as fast when operating alone). They both get repaired in 5 tu.



The system event log is shown in the figure above and is as follows:

1. At 100, Afails. It takes 5 $tu$to restore A.
2. From 100 to 105, $B$is operating alone and is experiencing a higher load.
3. At 115, Bfails. would normally be expected to fail at 120, however:
   a) From 0 to 100, it accumulated the equivalent of 100 $tu$of damage.
   b) From 100 to 105, it accumulated 10 $tu$of damage, which is twice the damage since it was operating alone. Put another way, Baged by 10 tuover a period of 5 tu.
   c) At 105, $A$is restored but Bhas only 10 $tu$of life remaining at this point.
   d) Bfails at 115.
4. At 120, $B$is repaired.
5. At 200, Afails again. Awould normally be expected to fail at 205; however, the failure of Bat 115 to 120 added additional damage to A. In other words, the age of Aat 115 was 10; by 120 it was 20. Thus it reached an age of 100 95 tulater at 200.
6. $A$is restored by 205.
7. At 235, Bfails. Bwould normally be expected to fail at 240; however, the failure of Aat 200 caused the reduction.
   a) At 200, Bhad an age of 80.
   b) By 205, Bhad an age of 90.

c) Bfails 30 tulater at 235.

8. The system itself never failed.

Additional Rules and Assumptions for Load Sharing Containers

1. The option of whether components operate while the system is down is defined at component level now (This is different from BlockSim 7, in which this option of the contained items inherit from container). Two rules here:

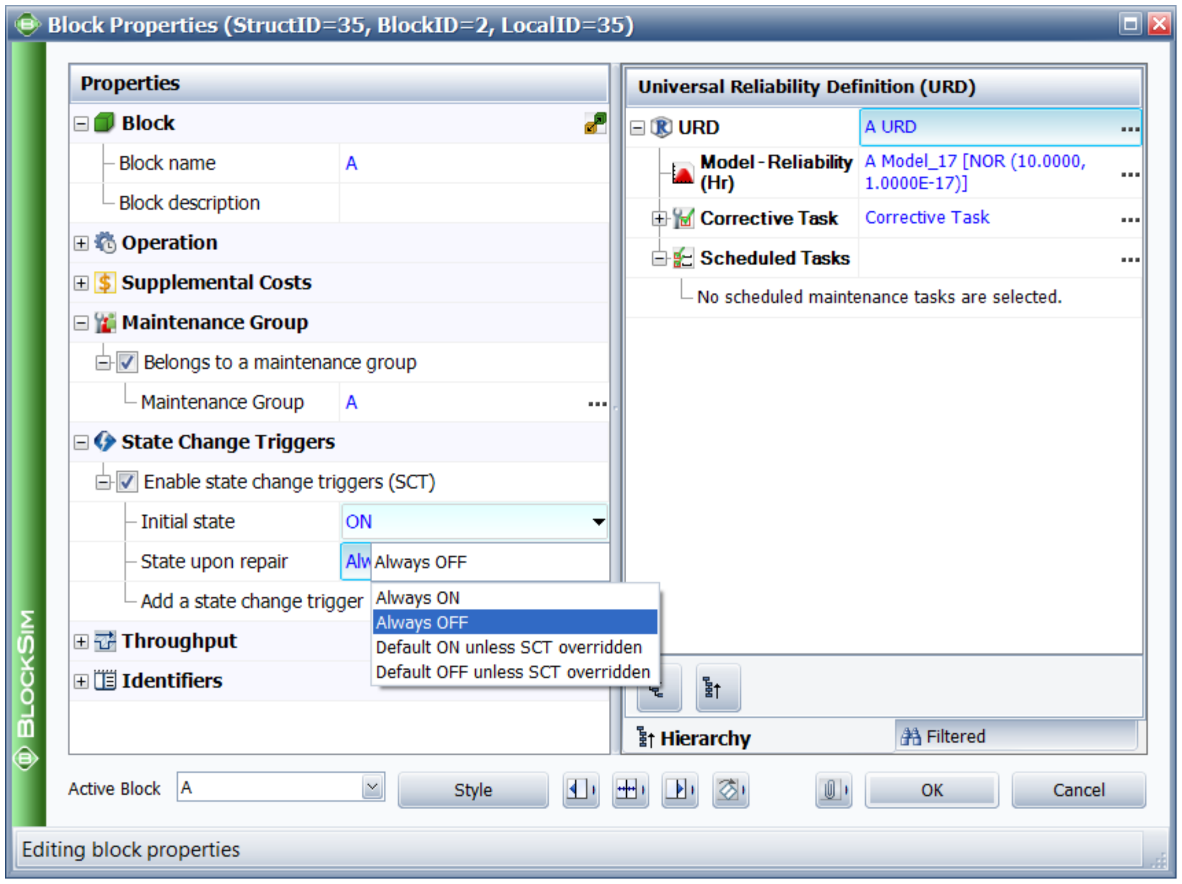
   a) If a path inside the container is down, blocks inside the container that are in that path do not continue to operate.

   b) Blocks that are up do not continue to operate while the container is down.

2. If a container brings the system down, the block that brought the container down is responsible for the system going down. (This is the opposite of standby containers.)

State Change Triggers

Consider a case where you have two generators, and one (A) is primary while the other (B) is standby. If A fails, you will turn B on. When A is repaired, it then becomes the standby. State change triggers (SCT) allow you to simulate this case. You can specify events that will activate and/or deactivate the block during simulation. The figure below shows the options for state change triggers in the Block Properties window.



Block Properties (StructID=35, BlockID=2, LocalID=35)

Properties

- Block
  - Block name: A
  - Block description
- Operation
- Supplemental Costs
- Maintenance Group
  - ☑ Belongs to a maintenance group
    - Maintenance Group: A
- State Change Triggers
  - ☑ Enable state change triggers (SCT)
    - Initial state: ON
    - State upon repair: Always OFF
    - Add a state change trigger
- Throughput
- Identifiers

Dropdown options: Always ON, Always OFF, Default ON unless SCT overridden, Default OFF unless SCT overridden

Universal Reliability Definition (URD)

- URD: A URD
  - Model - Reliability (Hr): A Model_17 [NOR (10.0000, 1.0000E-17)]
  - Corrective Task: Corrective Task
  - Scheduled Tasks
    - No scheduled maintenance tasks are selected.

Editing block properties

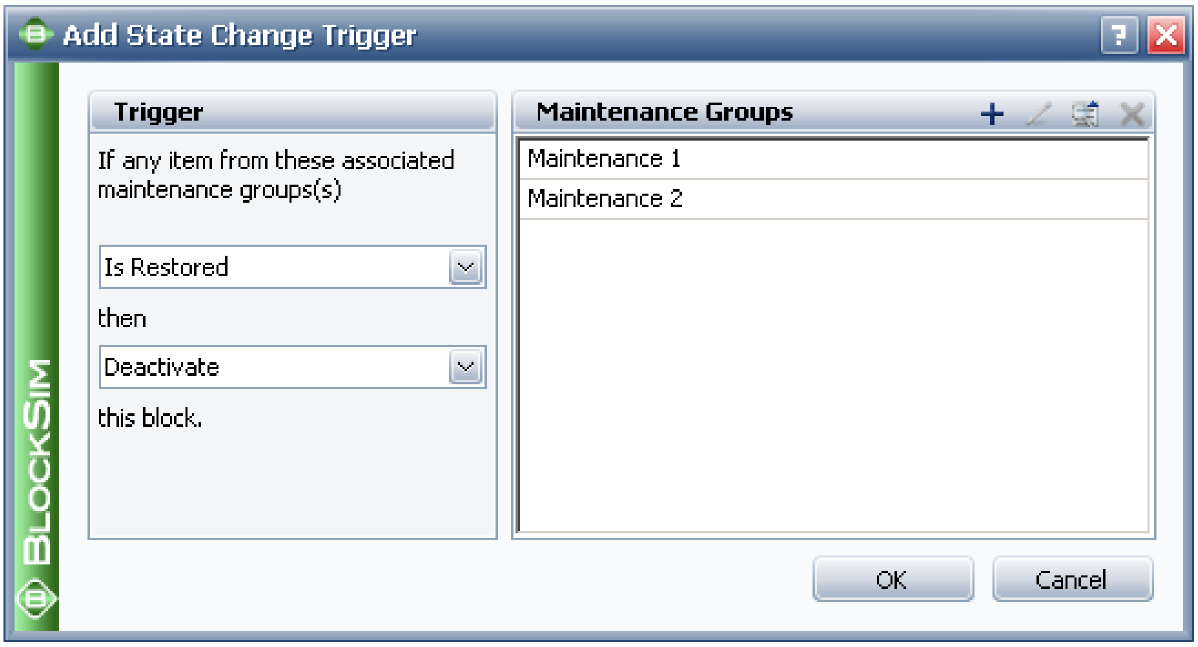
Once you have enabled state change triggers for a block, there are several options.

- **Initial state** allows you to specify the initial state for the block, either ON or OFF.
- **State upon repair** allows you to specify the state of the block after its repair. There are four choices: Always ON, Always OFF, Default ON unless SCT Overridden and Default OFF unless SCT Overridden. In the Assumptions

sections, we will explain what these choices mean and illustrate them using an example.
- **Add a state change trigger** allows you to add a state change trigger to the block.

The state change trigger can either activate or deactivate the block when items in specified maintenance groups go down or are restored. To define the state change trigger, specify the triggering event (i.e., an item goes down or an item is restored), the state change (i.e., the block is activated or deactivated) and the maintenance group(s) in which the triggering event must happen in order to trigger the state change. Note that the current block does not need to be part of the specified maintenance group(s) to use this functionality.

The State Change Trigger window is shown in the figure below:



Trigger

If any item from these associated maintenance groups(s)

Is Restored

then

Deactivate

this block.

Maintenance Groups

Maintenance 1

Maintenance 2

Assumptions

- A block cannot trigger events on itself. For example, if Block 1 is the only block that belongs to MG 1 and Block 1 is set to be turned ON or OFF based on MG 1, this trigger is ignored.
- OFF events cannot trigger other events. This means that things cannot be turned OFF in cascade. For example, if Block 1 going down turns OFF Block 2 and Block 2 going down turns OFF Block 3, a failure by Block 1 will not turn OFF Block 3. Block 3 would have to be directly associated with downing events of Block 1 for this to happen. The reason for this restriction is that allowing OFF events to trigger other events can cause circular reference problems. For example, four blocks A, B, C and D are in parallel. Block A belongs to MG A and initially it is ON. Block B belongs to MG B and its initial status is also ON. Block C belongs to MG C and its initial status is OFF. Block D belongs to MG D and its initial status is ON. A failure of Block A will turn OFF Block B. Then Block B will turn Block C ON and finally C will turn OFF Block D. However, if an OFF event for Block D will turn Block B ON, and an ON event for Block B will turn Block C OFF, and an OFF event for Block C will turn Block D ON, then there is a circular reference problem.
- Upon restoration states:
  - Always ON: Upon restoration, the block will always be on.
  - Always OFF: Upon restoration, the block will always be off.
  - Default ON unless SCT overridden: Upon restoration, the block will be on unless a request is made to turn this block off while the block is down and the request is still applicable at the time of restoration. For example, assume Block A's state upon repair is ON unless SCT overridden. If a failure of Block B triggers a request to turn Block A off but Block A is down, when the maintenance for Block A is completed, Block A will be

turned off if Block B is still down.

- Default off unless SCT overridden: Upon restoration, the block will be off unless a request is made to turn this block on while the block is down and the request is still applicable at the time of restoration

- Maintenance while block is off: Maintenance tasks will be performed. At the end of the maintenance, "upon restoration" rules will be checked to determine the state of the block.
- Assumptions for phases: The state of a block (on/off) will be determined at the beginning of each phase based on the "Initial state" setting of the block for that phase.
- If there are multiple triggering requests put on a block when it is down, only the latest one is considered. The latest request will cancel all requests before it. For example, Block A fails at 20 and is down until 70. Block B fails at 30 and Block C fails at 40. Block A has state change triggers enabled such that it will be activated when Block B fails and it will be deactivated when Block C fails. Thus from 20 to 70, at 30, Block B will put a request on Block A to turn it ON and at 40, Block C will put another request to turn it OFF. In this case, according to our assumption, the request from Block C at 40 will cancel the request from Block B at 30. In the end, only the request from Block C will be considered. Thus, Block A will be turned OFF at 70 when it is done with repair.

Example: Using SCT for Standby Rotation

This example illustrates the use of state change triggers in BlockSim (Version 8 and above) by using a simple standby configuration. *Note that this example could also be done using the standby container functionality in BlockSim.*

More specifically, the following settings are illustrated:

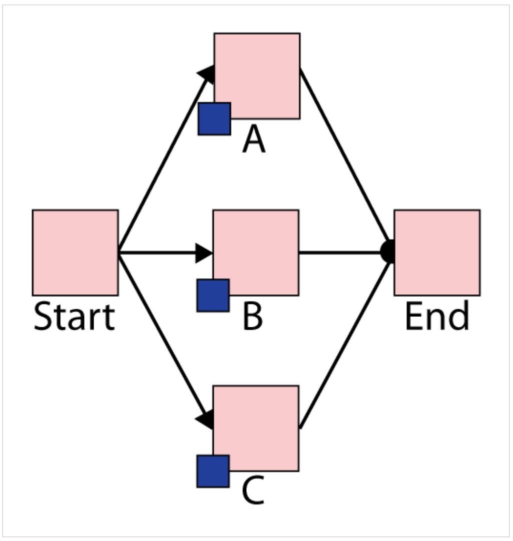
1. State Upon Repair: Default OFF unless SCT overridden
2. Activate a block if any item from these associated maintenance group(s) goes down

Problem Statement

Assume three devices A, B and C in a standby redundancy (or only one unit is needed for system operation). The system begins with device A working. When device A fails, B is turned on and repair actions are initiated on A. When B fails, C is turned on and so forth.

BlockSim Solution

The BlockSim model of this system is shown in the figure below.



- The failure distributions of all three blocks follow a Weibull distribution with Beta = 1.5 and Eta = 1,000 hours.
- The repair distributions of the three blocks follow a Weibull distribution with Beta = 1.5 and Eta = 100 hours.
- After repair, the blocks are "as good as new."

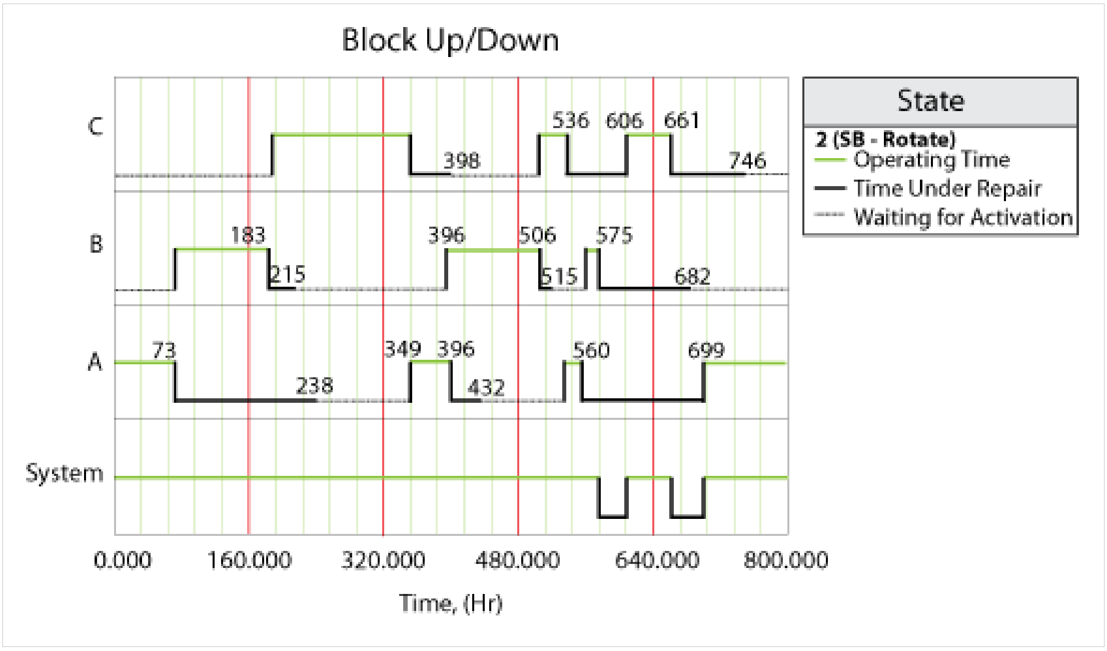
There are three maintenance groups, 2_A, 2_B and 2_C, set as follows:

- Block A belongs to maintenance group 2_A.
  - It has a state change trigger.
    - The initial state is ON and the state upon repair is "Default OFF unless SCT overridden."
    - If any item from maintenance group 2_C goes down, then activate this block.
- Block B belongs to maintenance group 2_B.
  - It has a state change trigger.
    - The initial state is OFF and the state upon repair is "Default OFF unless SCT overridden."
    - If any item from maintenance group 2_A goes down, then activate this block.
- Block C belongs to maintenance group 2_C.
  - It has a state change trigger.
    - The initial state is OFF and the state upon repair is "Default OFF unless SCT overridden."
    - If any item from maintenance group 2_B goes down, then activate this block.
- All blocks A, B and C are as good as new after repair.

System Events

The system event log for a single run through the simulation algorithm is shown in the Block Up/Down plot below, and is as follows:

1. At 73 hours, Block A fails and activates Block B.
2. At 183 hours, Block B fails and activates Block C.
3. At 215 hours, Block B is done with repair. At this time, Block C is operating, so according to the settings, Block B is standby.
4. At 238 hours, Block A is done with repair. At this time, Block C is operating. Thus Block A is standby.
5. At 349 hours, Block C fails and activates Block A.
6. At 396 hours, Block A fails and activates Block B.
7. At 398 hours, Block C is done with repair. At this time, Block B is operating. Thus Block C is standby.
8. At 432 hours, Block A is done with repair. At this time, Block B is operating. Thus Block A is standby.
9. At 506 hours, Block B fails and activates Block C.
10. At 515 hours, Block B is done with repair and stays standby because Block C is operating.
11. At 536 hours, Block C fails and activates Block A.
12. At 560 hours, Block A fails and activates Block B.
13. At 575 hours, Block B fails and makes a request to activate Block C. However, Block C is under repair at the time. Thus when Block C is done with repair at 606 hours, the OFF setting is overridden and it is operating immediately.
14. At 661 hours, Block C fails and makes a request to activate Block A. However, Block A is under repair at the time. Thus when Block A is done with repair at 699 hours, the OFF setting is overridden and it is operating immediately.
15. Block B and Block C are done with repair at 682 hours and at 746 hours respectively. However, at these two time points, Block A is operating. Thus they are both standby upon repair according to the settings.



See it in action...

More examples are available for using State Change Triggers (SCTs) in simulation diagrams. See also:

- Using SCT to Analyze Tire Maintenance
- Using SCT to Analyze Standby with Delay
- Using SCT to Model Two Standby Blocks
- SCT: The State Upon Repair Option
- Default OFF Unless SCT Overridden
- Default OFF Unless SCT Overridden

Discussion

Even though the examples and explanations presented here are deterministic, the sequence of events and logic used to view the system is the same as the one that would be used during simulation. The difference is that the process would be repeated multiple times during simulation and the results presented would be the average results over the multiple runs.

Additionally, multiple metrics and results are presented and defined in this chapter. Many of these results can also be used to obtain additional metrics not explicitly given in BlockSim's Simulation Results Explorer. As an example, to compute mean availability with inspections but without PMs, the explicit downtimes given for each event could be used. Furthermore, all of the results given are for operating times starting at zero to a specified end time (although the components themselves could have been defined with a non-zero starting age). Results for a starting time other than zero could be obtained by running two simulations and looking at the difference in the detailed results where applicable. As an example, the difference in uptimes and downtimes can be used to determine availabilities for a specific time window.

References

[1] http://www.weibull.com/hotwire/issue136/tooltips136.htm#units

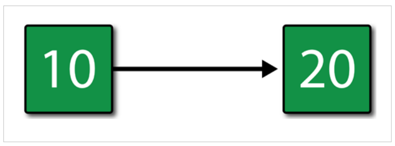
Chapter 8

Additional Analyses

The Repairable Systems Analysis Through Simulation chapter described the process of using discrete event simulation to perform basic system reliability, availability and maintainability analyses. This chapter discusses two additional types of analyses that can be performed with simulation: Throughput Analysis and Life Cycle Cost Analysis.

Throughput Analysis

In the prior sections, we concentrated on failure and repair actions of a system. In doing so, we viewed the system from a system/component up/down perspective. One could take this analysis a step further and consider system throughput as a part of the analysis. To define system throughput, assume that each component in the system processed (or made) something while operating. As an example, consider the case shown next with two components in series, each processing/producing 10 and 20 items per unit time (ipu) respectively.



In this case, the system configuration is not only the system's reliability-wise configuration but also its production/processing sequence. In other words, the first component processes/produces 10 ipu and the second component can process/produce up to 20 ipu. However, a block can only process/produce items it receives from the blocks before it. Therefore, the second component in this case is only receiving 10 ipu from the block before it. If we assume that neither component can fail, then the maximum items processed from this configuration would be 10 ipu. If the system were to operate for 100 time units, then the throughput of this system would be 1000 items, or $(100 \cdot 10)$.

Throughput Metrics and Terminology

In looking at throughput, one needs to define some terminology and metrics that will describe the behavior of the system and its components when doing such analyses. Some of the terminology used in BlockSim is given next.

- **System Throughput**: System throughput is the total amount of items processed/produced by the system over the defined period of time. In the two-component example, this is 1000items over 100time units.
- **Component Throughput**: The total amount of items processed or produced by each component (block). In the two-component example, this is 1000items each.
- **Component Maximum Capacity**: The maximum number of items that the component (block) could have processed/produced. This is simply the block's throughput rate multiplied by the run time. In the two-component example, this is $100 \cdot 10 = 1000$for the first component and $100 \cdot 20 = 2000$for the second component.
- **Component Uptime Capacity**: The maximum number of items the component could have processed/produced while it was up and running. In the two-component example, this is $1000$for the first component and $2000$for the second component, since we are assuming that the components cannot fail. If the components could fail, this number would be the component's uptime multiplied by its throughput rate.

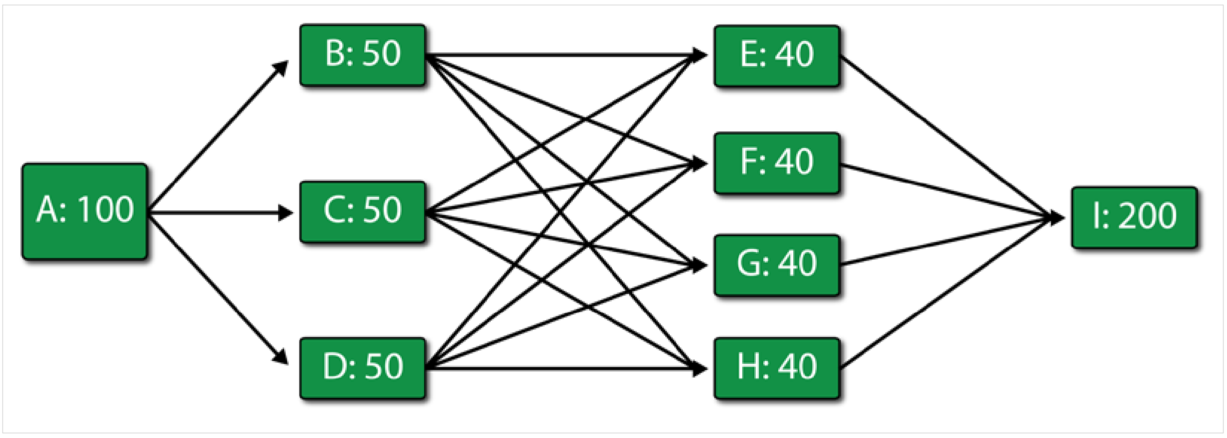
- **Component Excess Capacity**: The additional amount a component could have processed/produced while up and running. In the two-component example, this is $0$for the first component and $1000$for the second component.
- **Component Actual Utilization**: The ratio of the component throughput and the component maximum capacity. In the two-component example, this is $100\%$for the first component and $50\%$for the second component.
- **Component Uptime Utilization**: The ratio of the component throughput and the component uptime capacity. In the two-component example, this is $100\%$for the first component and $50\%$for the second component. Note that if the components had failed and experienced downtime, this number would be different for each component.
- **Backlog**: Items that the component could not process are kept in a backlog. Depending on the settings, a backlog may or may not be processed when an opportunity arises. The available backlog metrics include:
  - **Component Backlog**: The amount of a backlog present at the component at the end of the run (simulation).
  - **Component Processed Backlog**: The amount of backlog processed by the component.
  - **Excess Backlog**: Under specific settings in BlockSim, components can accept only a limited backlog. In these cases, a backlog that was rejected is stored in the Excess Backlog category.

Overview of Throughput Analysis

To examine throughput, consider the following scenarios.

Scenario 1

Consider the system shown in the figure below.



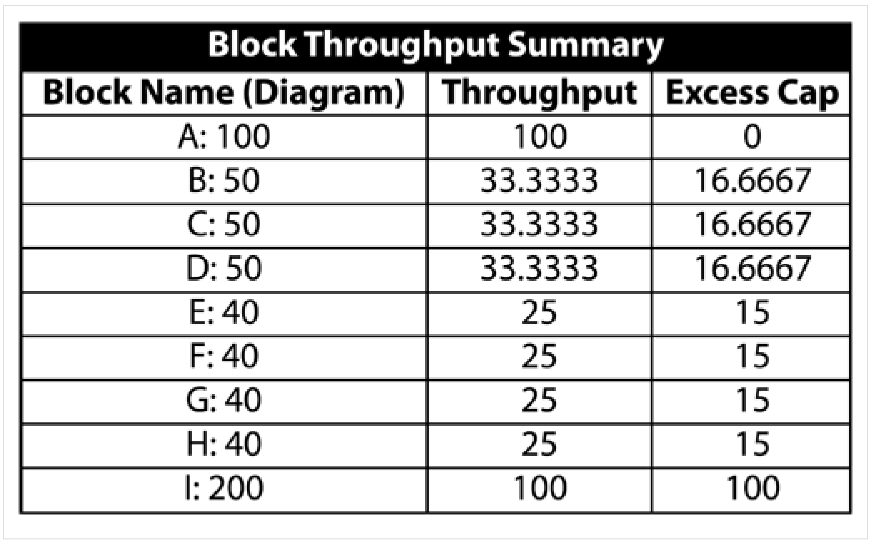
Blocks Athrough Iproduce a number of items per unit time as identified next to each letter (e.g., A: 100 implies 100items per time unit for A). The connections shown in the RBD show the physical path of the items through the process (or production line). For the sake of simplicity, also assume that the blocks can never fail and that items are routed equally to each path.

This then implies that the following occurs over a single time unit:

- Unit Amakes 100 items and routes 33.33 to B, 33.33 to $C$and 33.33 to D.
- In turn, B, $C$and Droute their 33.33 to E, F, $G$and H(8.33 to each path).
- E, F, $G$and Hroute 25 each to I.
- Iprocesses all 100.
- The system produces 100 items.

Thus, the following table would represent the throughput and excess capacity of each block after one time unit.

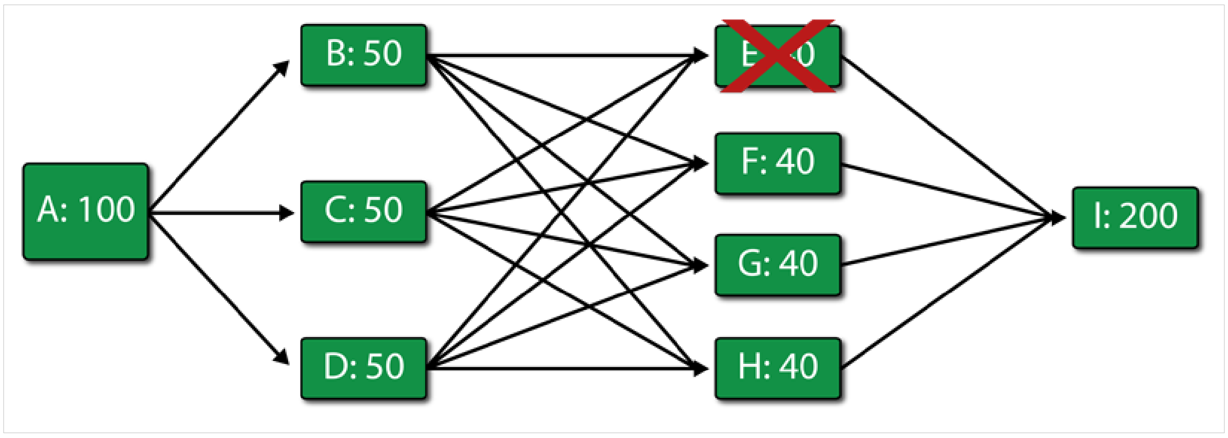
Run summary for Scenario 1.



Block Throughput Summary		
Block Name (Diagram)	**Throughput**	**Excess Cap**
A: 100	100	0
B: 50	33.3333	16.6667
C: 50	33.3333	16.6667
D: 50	33.3333	16.6667
E: 40	25	15
F: 40	25	15
G: 40	25	15
H: 40	25	15
I: 200	100	100

Scenario 2

Now consider the figure below where it is assumed that block E has failed.

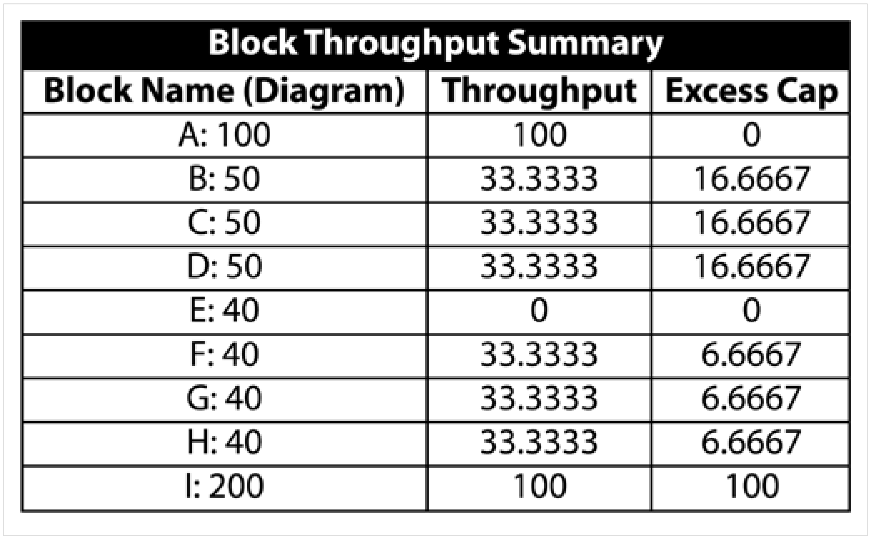


Then:

- Unit A makes 100 items and routes 33.33 to B, 33.33 to C and 33.33 to D.
- In turn, B, C and D route their 33.33 to F, G and H (11.11 to each path that has an operating block at the end).
- F, G and H route 33.33 each to I.
- I processes all 100.
- he system produces 100 items.

A summary result table is shown next:

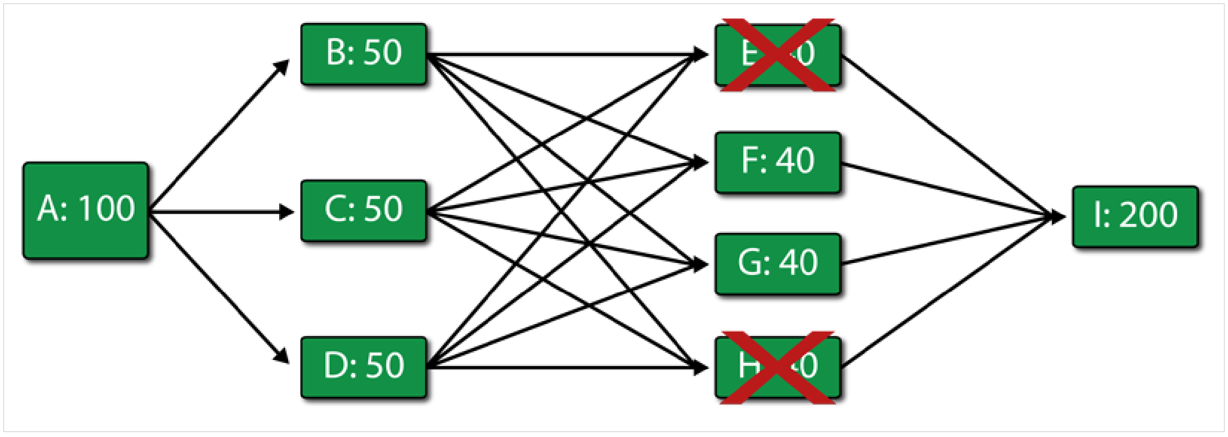
Run summary for Scenario 2.



Block Throughput Summary		
Block Name (Diagram)	**Throughput**	**Excess Cap**
A: 100	100	0
B: 50	33.3333	16.6667
C: 50	33.3333	16.6667
D: 50	33.3333	16.6667
E: 40	0	0
F: 40	33.3333	6.6667
G: 40	33.3333	6.6667
H: 40	33.3333	6.6667
I: 200	100	100

Scenario 3

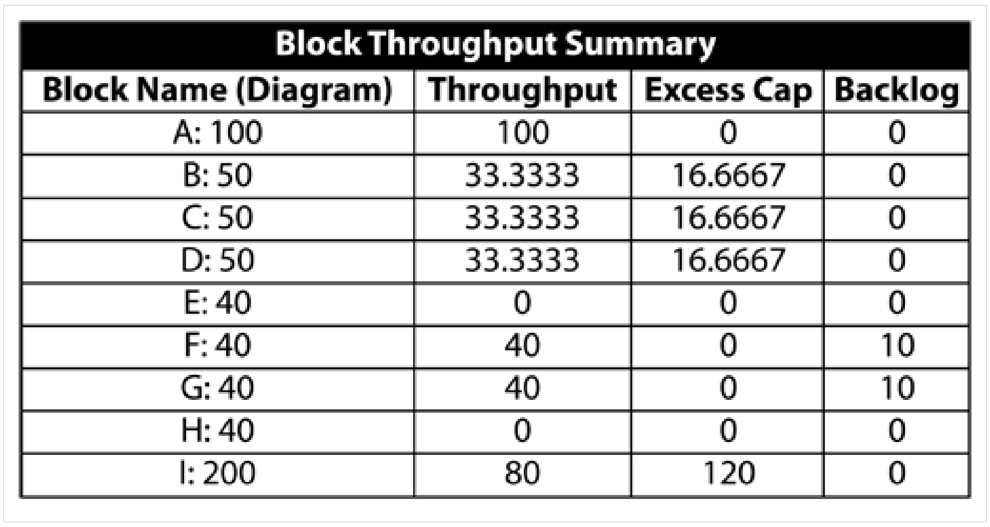
Finally, consider the figure below where both $E$and Hhave failed.



Then:

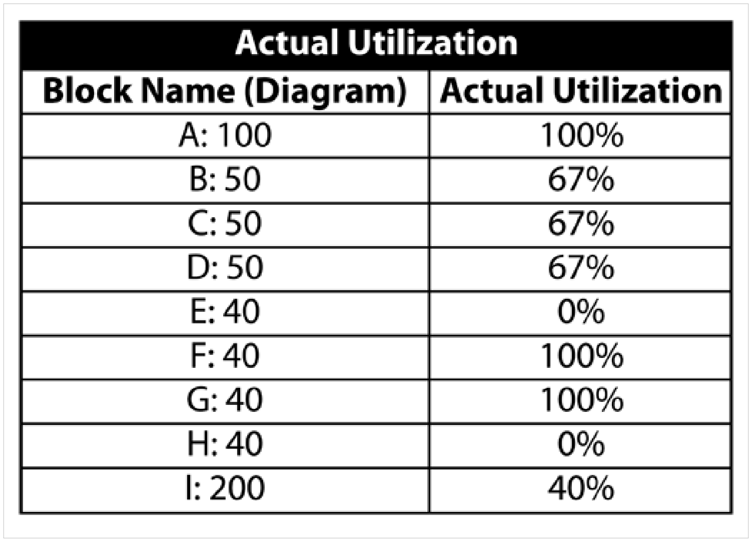
- Unit Amakes 100 items and routes 33.33 to B, 33.33 to $C$and 33.33 to D.
- In turn, B, $C$and Droute their 33.33 to $F$and G(16.66 to each path that has an operating block at the end).
- $F$and Gget 50 items each.
- $F$and Gprocess and route 40 each (their maximum processing capacity) to I. Both have a backlog of 10 since they could not process all 50 items they received.
- Iprocesses all 80.
- The system produces 80 items.

Run summary for Scenario 3.



Block Throughput Summary			
Block Name (Diagram)	Throughput	Excess Cap	Backlog
A: 100	100	0	0
B: 50	33.3333	16.6667	0
C: 50	33.3333	16.6667	0
D: 50	33.3333	16.6667	0
E: 40	0	0	0
F: 40	40	0	10
G: 40	40	0	10
H: 40	0	0	0
I: 200	80	120	0

Utilization summary for Scenario 3.



Actual Utilization	
Block Name (Diagram)	Actual Utilization
A: 100	100%
B: 50	67%
C: 50	67%
D: 50	67%
E: 40	0%
F: 40	100%
G: 40	100%
H: 40	0%
I: 200	40%

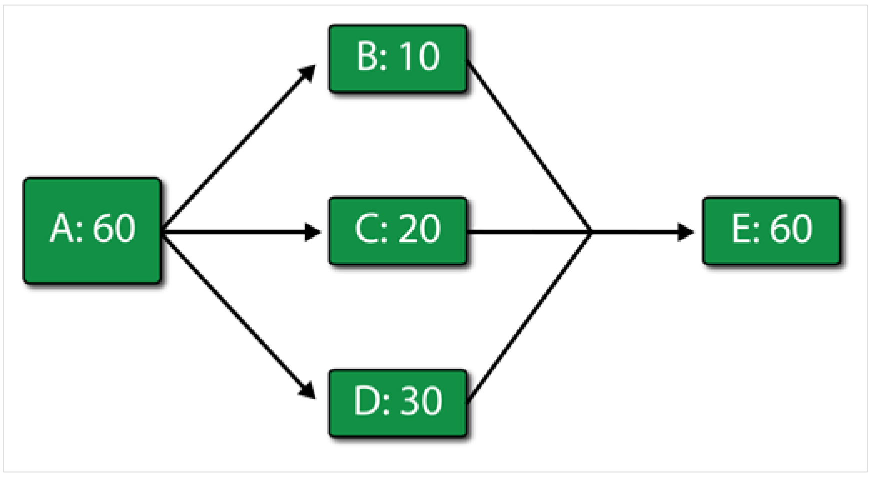
It can be easily seen that the bottlenecks in the system are the blocks $F$and G.

Throughput Analysis Options

In BlockSim, specific throughput properties can be set for the blocks in the diagram.

Throughput: The number of items that the block can process per unit time.

Allocation: Specify the allocation scheme across multiple paths (i.e., equal or weighted). This option is shown in the figure below.



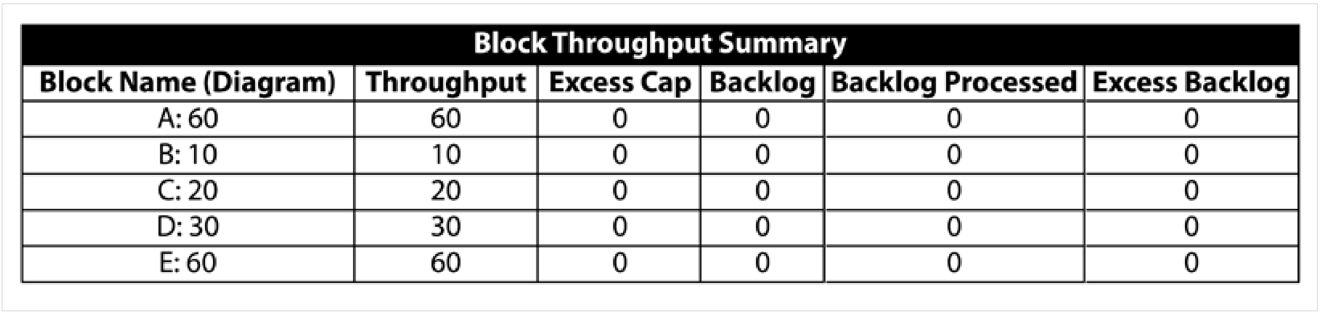
To explain these settings, consider the example shown in the above figure, which uses the same notation as before.

If the **Weighted allocation across paths** option is chosen, then the 60 items made by Awill be allocated to B, C and Dbased on their throughput capabilities. Specifically, the portion that each block will receive, P_i, is:

$$P_i = \frac{Throughput_i}{\sum\limits_{j=1}^{N} Throughput_j} \quad (eqn1)$$

The actual amount is then the (portion ·available units). In this case, the portion allocated to $B$is $\frac{10}{60}$;the portion allocated to $C$is $\frac{20}{60}$and the portion allocated to $D$is $\frac{30}{60}$. When a total of 60 units is processed through A, Bwill get 10, Cwill get 20 and Dwill get 30.
The results would then be as shown in the table below.

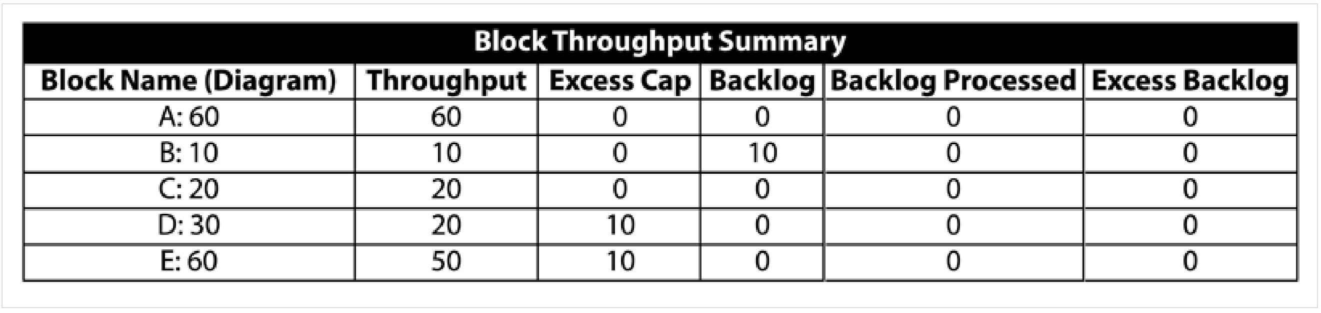
Throughput summary using weighted allocation across paths.



Block Throughput Summary					
Block Name (Diagram)	**Throughput**	**Excess Cap**	**Backlog**	**Backlog Processed**	**Excess Backlog**
A: 60	60	0	0	0	0
B: 10	10	0	0	0	0
C: 20	20	0	0	0	0
D: 30	30	0	0	0	0
E: 60	60	0	0	0	0

If the **Allocate equal share to all paths** option is chosen, then 20 units will be sent to B, $C$and Dregardless of their processing capacity, yielding the results shown in the table below.

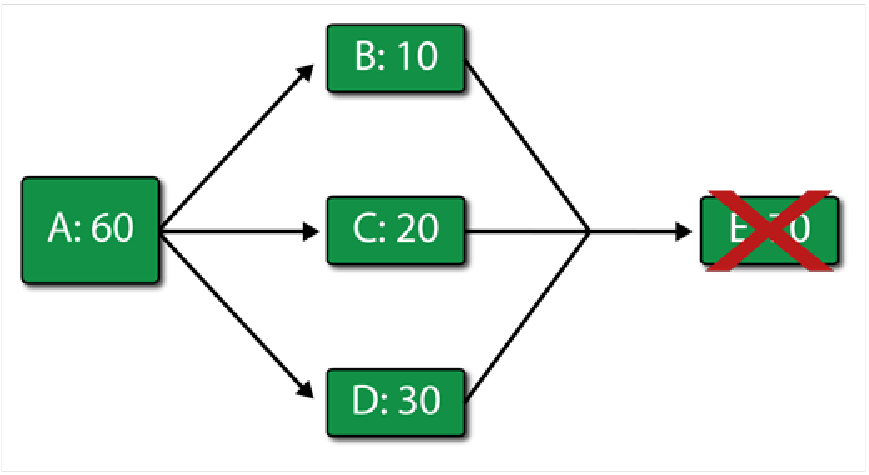
Throughput summary using an equal allocation across paths.



Block Throughput Summary					
Block Name (Diagram)	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A: 60	60	0	0	0	0
B: 10	10	0	10	0	0
C: 20	20	0	0	0	0
D: 30	20	10	0	0	0
E: 60	50	10	0	0	0

Send units to failed blocks: Decide whether items should be sent to failed parts. If this option is not selected, the throughput units are allocated only to operational units. Otherwise, if this option is selected, units are also allocated to failed blocks and they become part of the failed block's backlog.

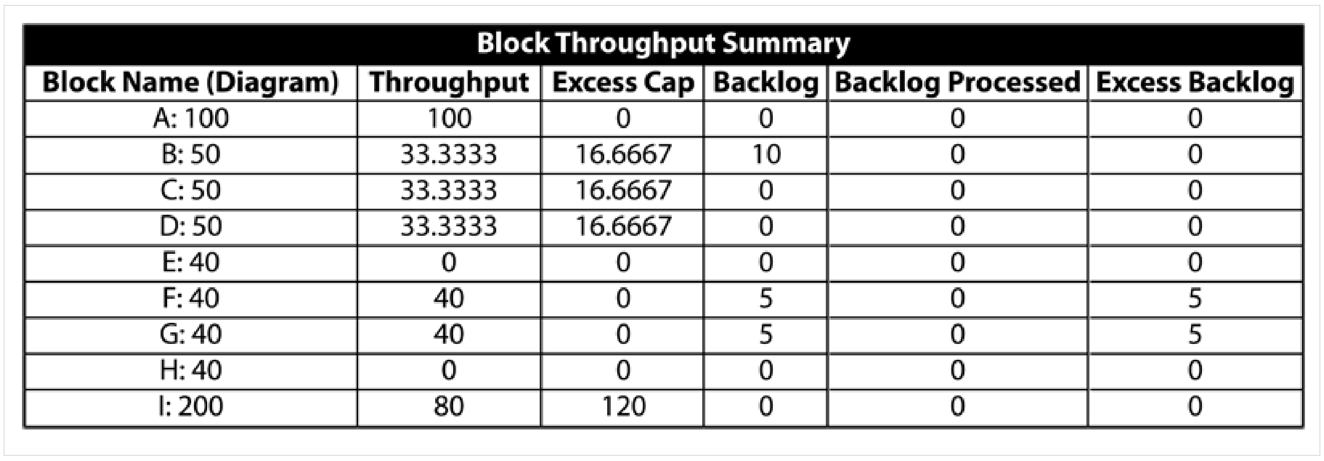
In the special case in which one or more blocks fail, causing a disruption in the path, and the **Send units to failed blocks** option is not selected, then the blocks that have a path to the failed block(s) will not be able to process any items, given the fact that they cannot be sent to the forward block. In this case, these blocks will keep all items received in their backlog. As an example, and using the figure below, if $E$is failed (and Ecannot accept items in its backlog while failed), then $B,C$and Dcannot forward any items to it. Thus, they will not process any items sent to them from A. Items sent from Awill be placed in the backlogs of items $B,C$and D.



Process/Ignore backlog: Identify how a block handles backlog. A block can ignore or process backlog. Items that cannot be processed are kept in a backlog bin and are processed as needed.

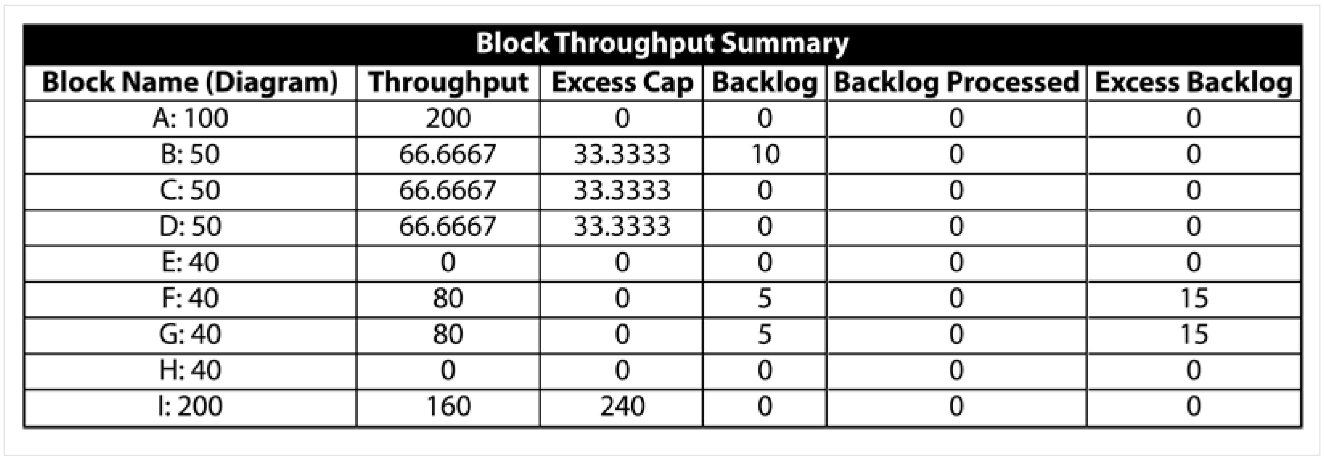
Additionally, you can set the maximum number of items that can be stored in the backlog. When you choose to ignore the backlog, BlockSim will still report items that cannot be processed in the backlog column. However, it will let the backlog accumulate and never process it. In the case of a limited backlog, BlockSim will not accumulate more backlog than the maximum allowed and will discard all items sent to the block if they exceed its backlog capacity. It will keep count of the units that did not make it in the backlog in a category called Excess Backlog. To illustrate this, reconsider Scenario 3, but with both $F$and Ghaving a limited backlog of 5. After a single time unit of operation, the results would be as shown in the tables below.

Scenario 3 summary with *F* and *G* having a limited blacklog and after one time unit.



Block Throughput Summary					
Block Name (Diagram)	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A: 100	100	0	0	0	0
B: 50	33.3333	16.6667	10	0	0
C: 50	33.3333	16.6667	0	0	0
D: 50	33.3333	16.6667	0	0	0
E: 40	0	0	0	0	0
F: 40	40	0	5	0	5
G: 40	40	0	5	0	5
H: 40	0	0	0	0	0
I: 200	80	120	0	0	0

Scenario 3 summary with *F* and *G* having a limited backlog and after two time units.



Block Throughput Summary					
Block Name (Diagram)	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A: 100	200	0	0	0	0
B: 50	66.6667	33.3333	10	0	0
C: 50	66.6667	33.3333	0	0	0
D: 50	66.6667	33.3333	0	0	0
E: 40	0	0	0	0	0
F: 40	80	0	5	0	15
G: 40	80	0	5	0	15
H: 40	0	0	0	0	0
I: 200	160	240	0	0	0

Note that the blocks will never be able to process the backlog in this example. However, if we were to observe the system for a longer operation time and through failures and repairs of the other blocks, there would be opportunities for the blocks to process their backlogs and catch up. It is very important to note that when simulating a system with failures and repairs in BlockSim, you must define the block as one that operates through system failure if you wish for backlog to be processed. If this option is not set, the block will not operate through system failure and thus will not be able to process any backlog items when components that cause system failure (from an RBD perspective) fail.

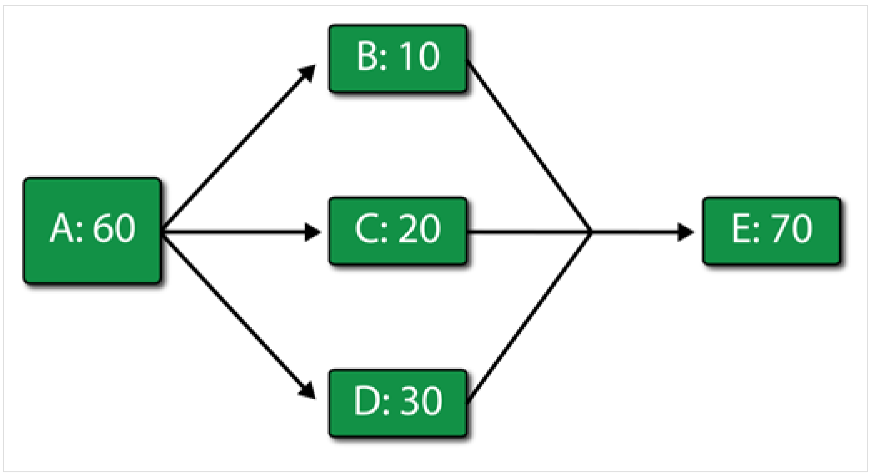
Variable Throughput

In many real-world cases throughput can change over time (i.e., throughput through a single component is not a constant but a function of time). The discussion in this chapter is devoted to cases of constant, non-variable throughput. BlockSim does model variable throughput using phase diagrams. These are discussed in Introduction to Reliability Phase Diagrams.

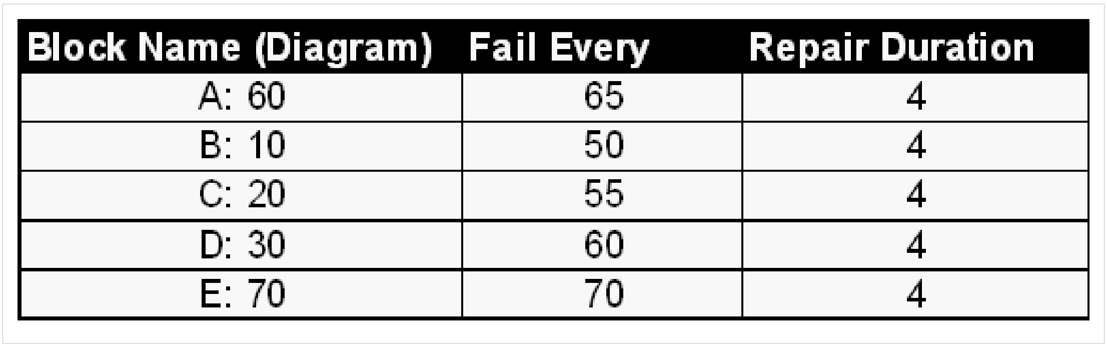
A Simple Throughput Analysis Example

The prior sections illustrated the basics concepts in throughput analysis. However, they did not take into account the reliability and maintenance properties of the blocks and the system. In a complete analysis, these would also need to be incorporated. The following simple example incorporates failures and repairs. Even though the example is trivial, the concepts presented here form the basis of throughput analysis in BlockSim. The principles remain the same no matter how complex the system.

Consider the simple system shown in the figure below, but with Eoperating.



In addition, consider the following deterministic failure and repair characteristics:



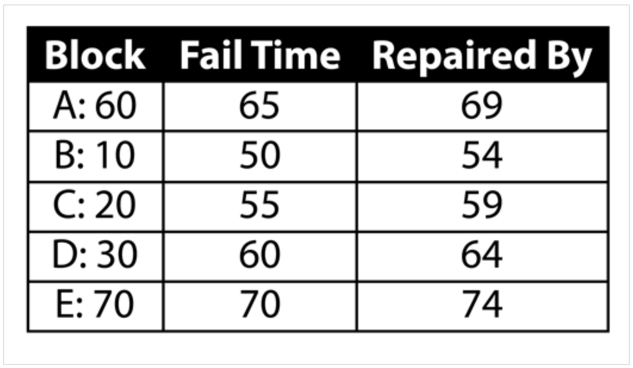
Block Name (Diagram)	Fail Every	Repair Duration
A: 60	65	4
B: 10	50	4
C: 20	55	4
D: 30	60	4
E: 70	70	4

Also:

- Set all units to operate through system failure.
- Do not add spare part pools or crews (use defaults).
- Do not send items to failed units.
- Use a weighted allocation scheme.

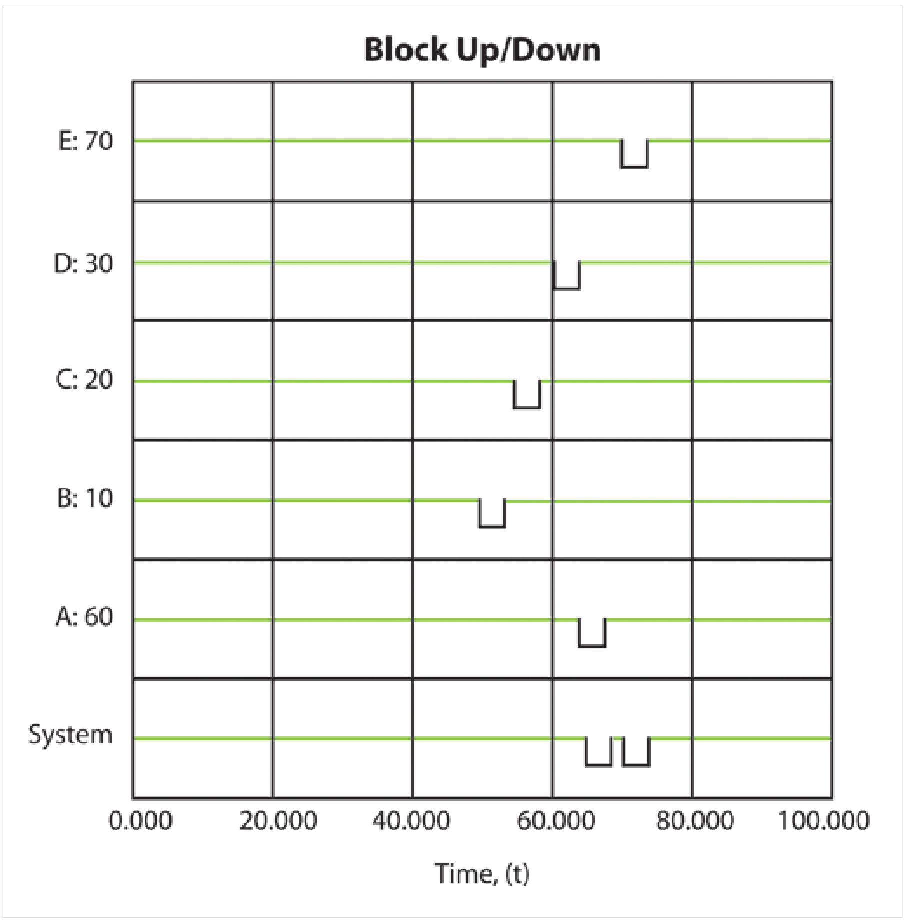
Event History: 0 to 100 Time Units

Then the system behavior from 0 to 100 time units is given in the table below. The system event history is as follows:



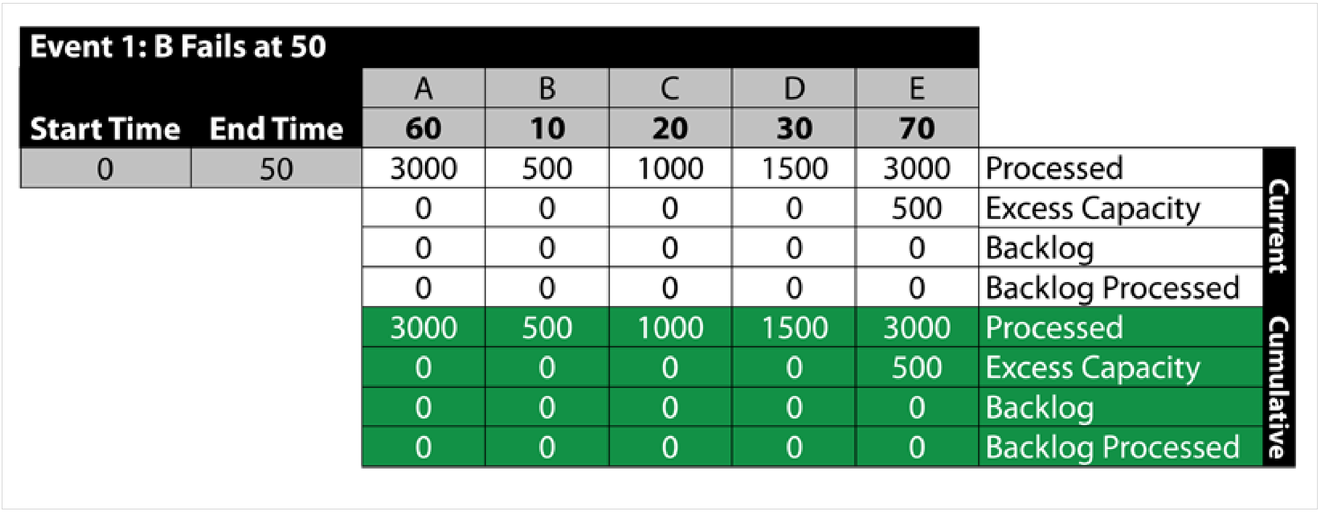
Block	Fail Time	Repaired By
A: 60	65	69
B: 10	50	54
C: 20	55	59
D: 30	60	64
E: 70	70	74

Once the system history has been established, we can examine the throughput behavior of this system from 0 to 100 by observing the sequence of events and their subsequent effect on system throughput.



Event 1: B Fails at 50

- At 50, B fails.
- From 0 to 50, A processes $50 \cdot 60 = 3000$items.
- 500 are sent to B , 1000 to $C$and 1500 to D. There is no excess capacity at B , Cor D.
- B , $C$and Dprocess and send 3000 items to E. Because the capacity of $E$is 3500, Enow has an excess capacity of 500.
- The next table summarizes these results:

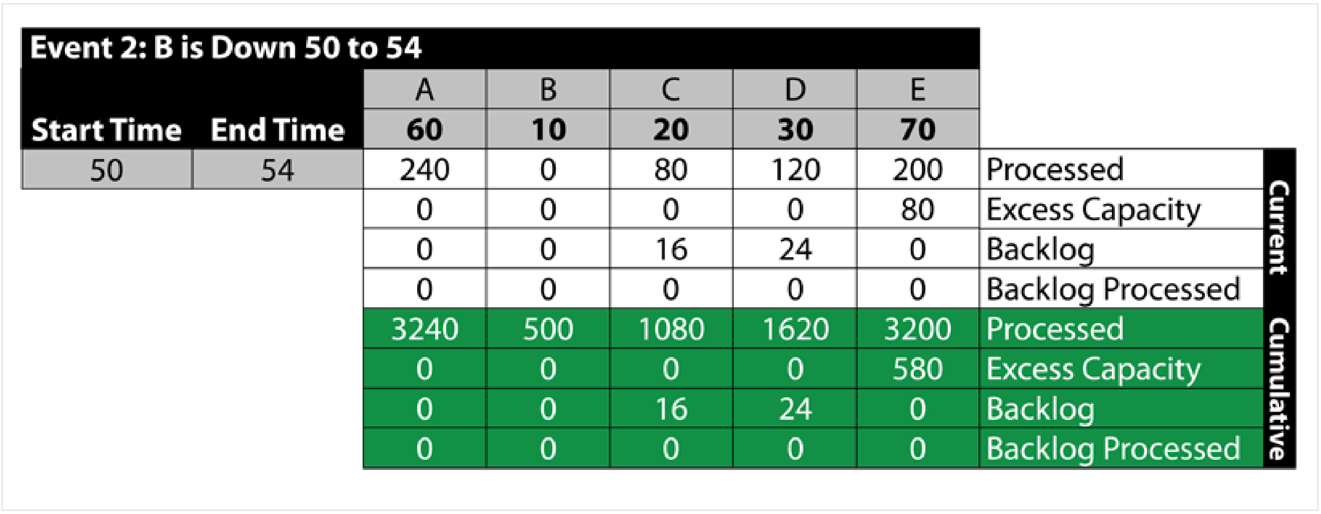


Event 1: B Fails at 50								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
0	50	3000	500	1000	1500	3000	Processed	Current
		0	0	0	0	500	Excess Capacity	
		0	0	0	0	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		3000	500	1000	1500	3000	Processed	Cumulative
		0	0	0	0	500	Excess Capacity	
		0	0	0	0	0	Backlog	
		0	0	0	0	0	Backlog Processed	

Event 2: B is Down 50 to 54

- From 50 to 54, B is down.

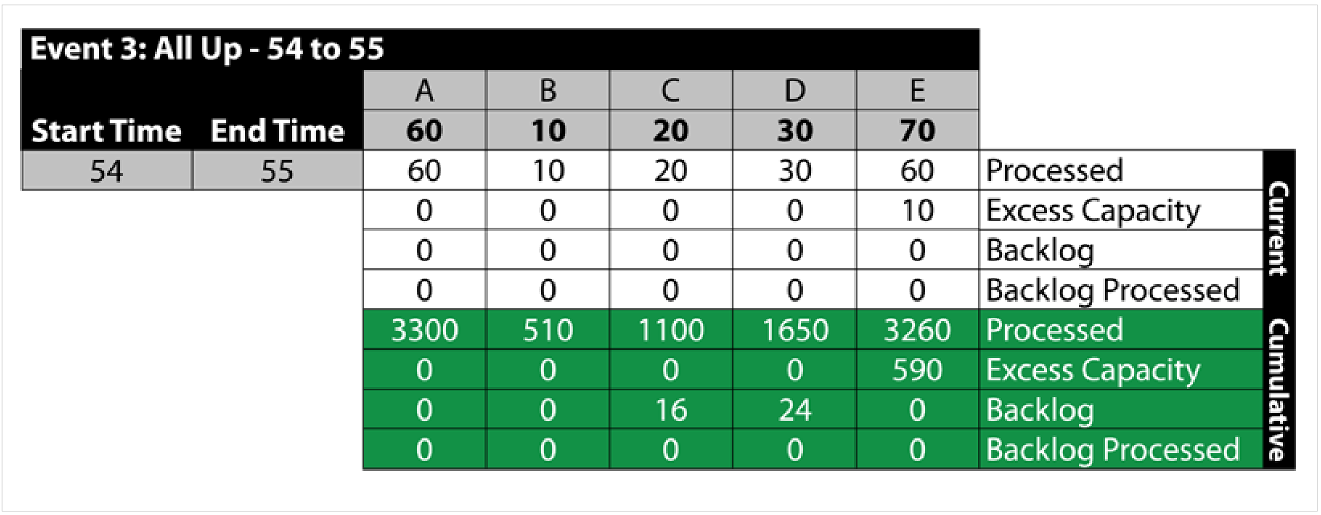
- A processes 240 items and sends 96 to C and 144 to D.
- D and C can only process 80 and 120 respectively during this time. Thus, they get backlogs of 16 and 24 respectively.
- The 200 processed are sent to E. E has an excess capacity of 80 during this time period.
- The next table summarizes these results:



Event 2: B is Down 50 to 54								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
50	54	240	0	80	120	200	Processed	Current
		0	0	0	0	80	Excess Capacity	
		0	0	16	24	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		3240	500	1080	1620	3200	Processed	Cumulative
		0	0	0	0	580	Excess Capacity	
		0	0	16	24	0	Backlog	
		0	0	0	0	0	Backlog Processed	

Event 3: All Up 54 to 55

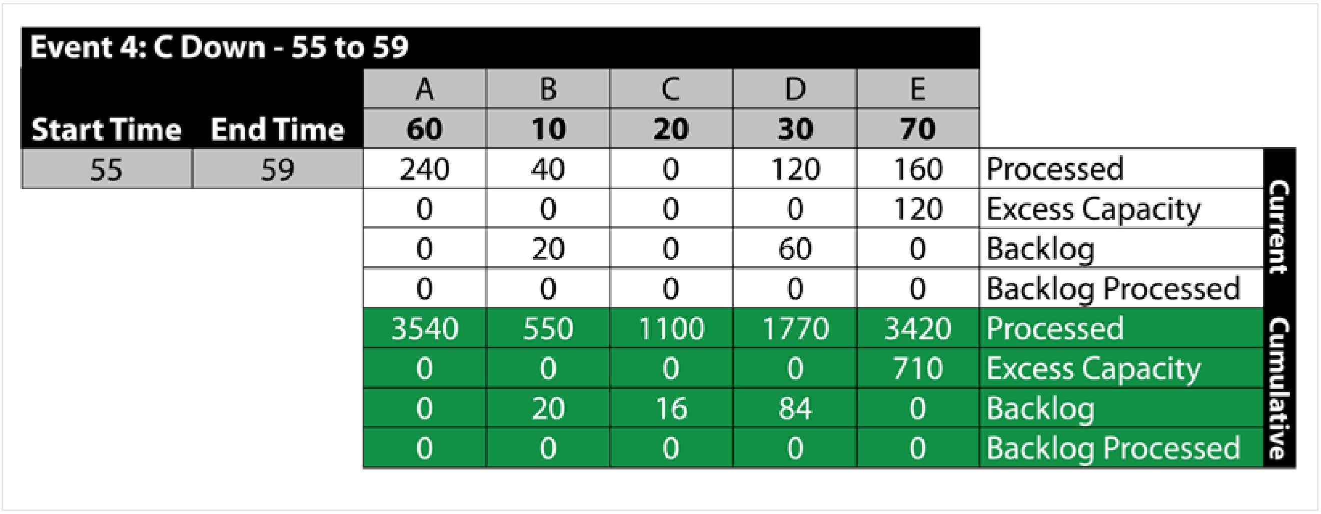
The next table summarizes the results:



Event 3: All Up - 54 to 55								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
54	55	60	10	20	30	60	Processed	Current
		0	0	0	0	10	Excess Capacity	
		0	0	0	0	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		3300	510	1100	1650	3260	Processed	Cumulative
		0	0	0	0	590	Excess Capacity	
		0	0	16	24	0	Backlog	
		0	0	0	0	0	Backlog Processed	

Event 4: C is Down 55 to 59

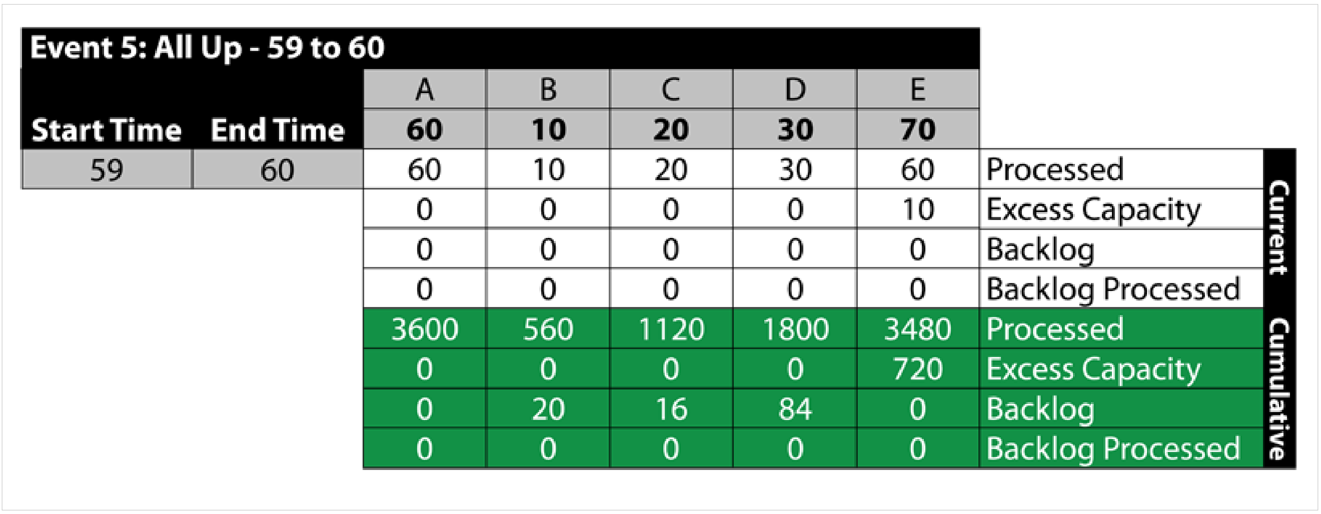
The next table summarizes the results:



Event 4: C Down - 55 to 59								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
55	59	240	40	0	120	160	Processed	Current
		0	0	0	0	120	Excess Capacity	
		0	20	0	60	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		3540	550	1100	1770	3420	Processed	Cumulative
		0	0	0	0	710	Excess Capacity	
		0	20	16	84	0	Backlog	
		0	0	0	0	0	Backlog Processed	

Event 5: All Up 59 to 60

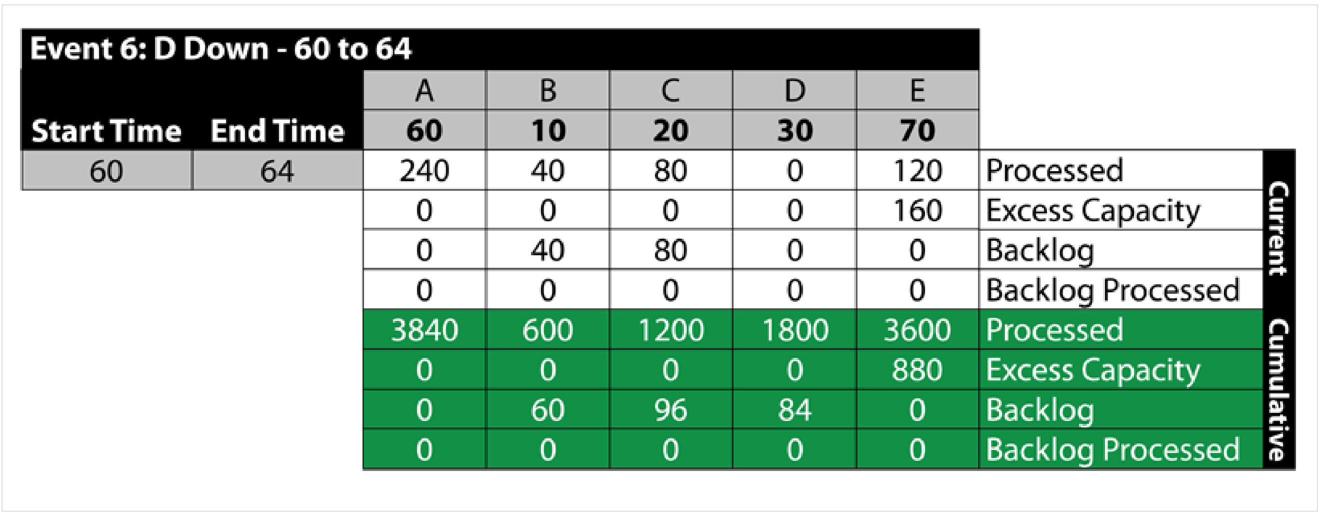
The next table summarizes the results:



Event 5: All Up - 59 to 60								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
59	60	60	10	20	30	60	Processed	Current
		0	0	0	0	10	Excess Capacity	
		0	0	0	0	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		3600	560	1120	1800	3480	Processed	Cumulative
		0	0	0	0	720	Excess Capacity	
		0	20	16	84	0	Backlog	
		0	0	0	0	0	Backlog Processed	

Event 6: *D*is Down 60 to 64

The next table summarizes the results:



Event 6: D Down - 60 to 64								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
60	64	240	40	80	0	120	Processed	Current
		0	0	0	0	160	Excess Capacity	
		0	40	80	0	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		3840	600	1200	1800	3600	Processed	Cumulative
		0	0	0	0	880	Excess Capacity	
		0	60	96	84	0	Backlog	
		0	0	0	0	0	Backlog Processed	

Event 7: All Up 64 to 65

The next table summarizes the results:

Event 7: All Up - 64 to 65								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
64	65	60	10	20	30	60	Processed	Current
		0	0	0	0	10	Excess Capacity	
		0	0	0	0	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		3900	610	1220	1830	3660	Processed	Cumulative
		0	0	0	0	890	Excess Capacity	
		0	60	96	84	0	Backlog	
		0	0	0	0	0	Backlog Processed	

Event 8: A is Down 65 to 69

Between 65 and 69, A fails. This stops the flow of items in the system and provides an opportunity for the other blocks to process their backlogs. As an example, B processes 40 items from the 60 items in its backlog. Specifically:

Event 8: A Down - 65 to 69								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
65	69	0	40	80	84	204	Processed	Current
		0	0	0	36	76	Excess Capacity	
		0	-40	-80	-84	0	Backlog	
		0	40	80	84	0	Backlog Processed	
		3900	650	1300	1914	3864	Processed	Cumulative
		0	0	0	36	966	Excess Capacity	
		0	20	16	0	0	Backlog	
		0	40	80	84	0	Backlog Processed	

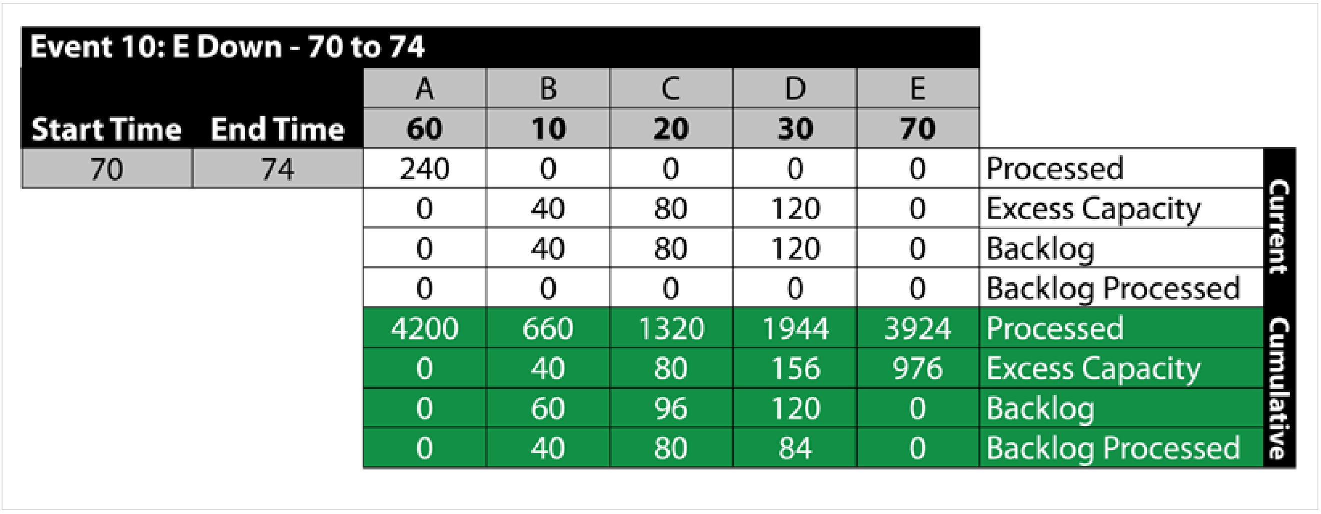
Event 9: All Up 69 to 70

The next table summarizes the results:

Event 9: All Up - 69 to 70								
		A	B	C	D	E		
Start Time	End Time	60	10	20	30	70		
69	70	60	10	20	30	60	Processed	Current
		0	0	0	0	10	Excess Capacity	
		0	0	0	0	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		3960	660	1320	1944	3924	Processed	Cumulative
		0	0	0	36	976	Excess Capacity	
		0	20	16	0	0	Backlog	
		0	40	80	84	0	Backlog Processed	

Event 10: $E$is Down 60 to 64

From 70 to 74, $E$is down. Because we specified that we will not send items to failed units, B , $C$and Dreceive items from A but they do not process them, since processing would require that items be sent to E. The items received by B , $C$and $D$are added to their respective backlogs. Furthermore, since they could have processed them if Ehad been up, all three blocks have an excess capacity for this period. Specifically:



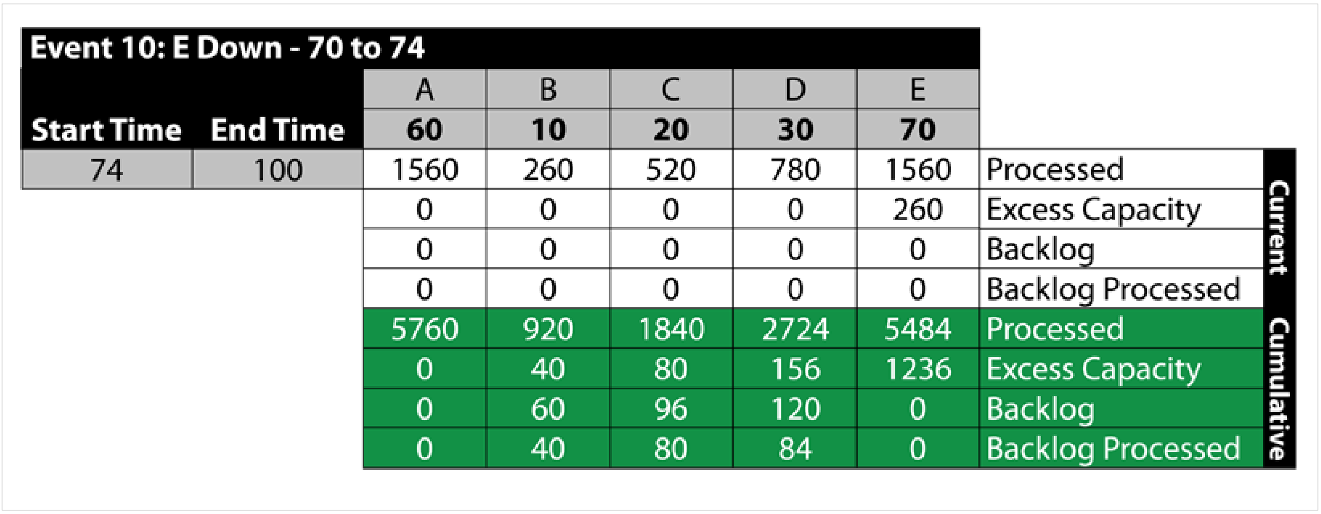
Event 10: E Down - 70 to 74

Start Time	End Time	A 60	B 10	C 20	D 30	E 70		
70	74	240	0	0	0	0	Processed	Current
		0	40	80	120	0	Excess Capacity	
		0	40	80	120	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		4200	660	1320	1944	3924	Processed	Cumulative
		0	40	80	156	976	Excess Capacity	
		0	60	96	120	0	Backlog	
		0	40	80	84	0	Backlog Processed	

It should be noted that if we had allowed items to be sent to failed blocks, B , $C$and Dwould have processed the items received and the backlog would have been at E. The rest of the time, all units are up.

Event 11: All Up 74 to 100

The next table summarizes the results:

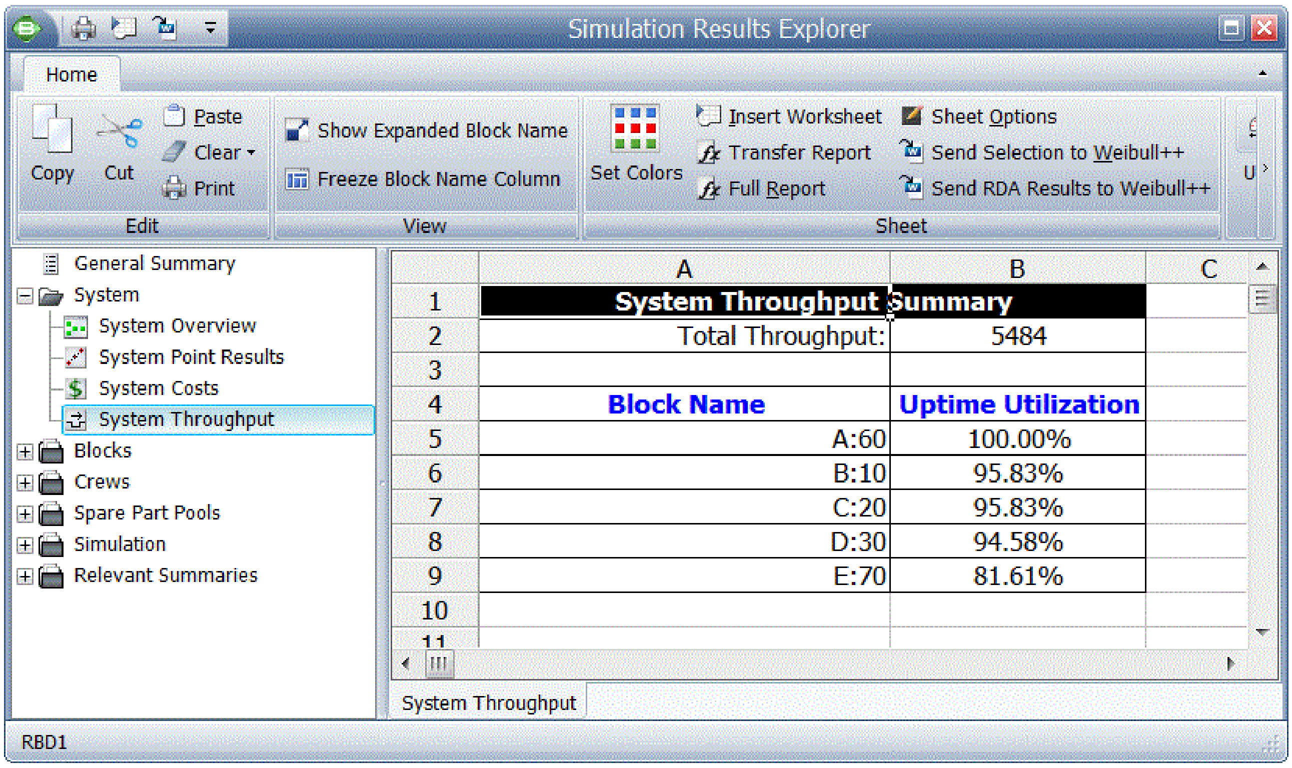


Event 10: E Down - 70 to 74

Start Time	End Time	A 60	B 10	C 20	D 30	E 70		
74	100	1560	260	520	780	1560	Processed	Current
		0	0	0	0	260	Excess Capacity	
		0	0	0	0	0	Backlog	
		0	0	0	0	0	Backlog Processed	
		5760	920	1840	2724	5484	Processed	Cumulative
		0	40	80	156	1236	Excess Capacity	
		0	60	96	120	0	Backlog	
		0	40	80	84	0	Backlog Processed	

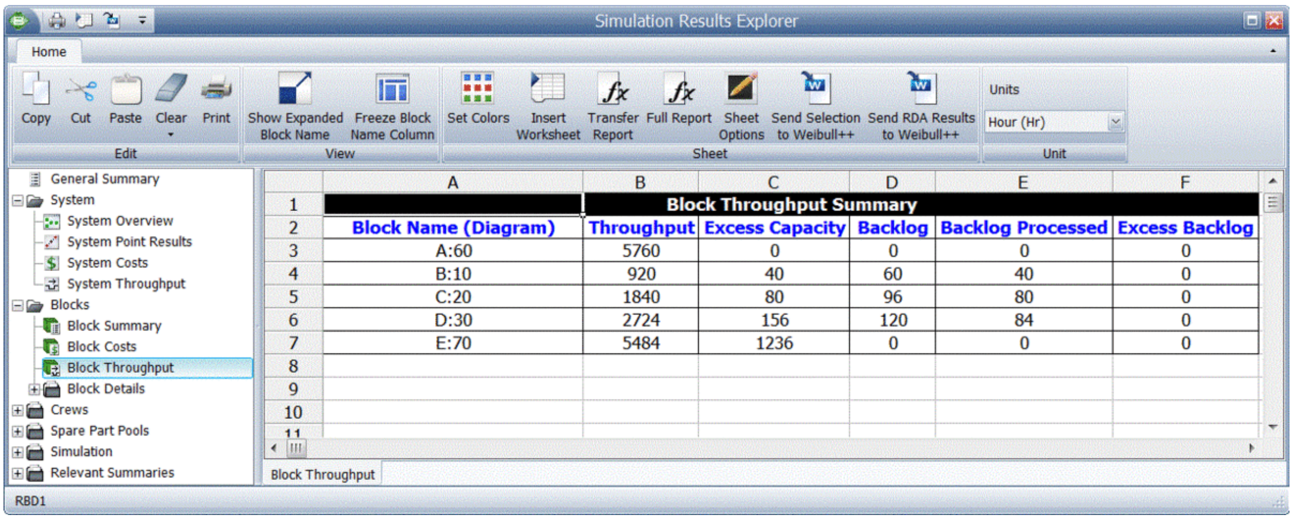
Exploring the Results

BlockSim provides all of these results via the Simulation Results Explorer. The figure below shows the system throughput summary.



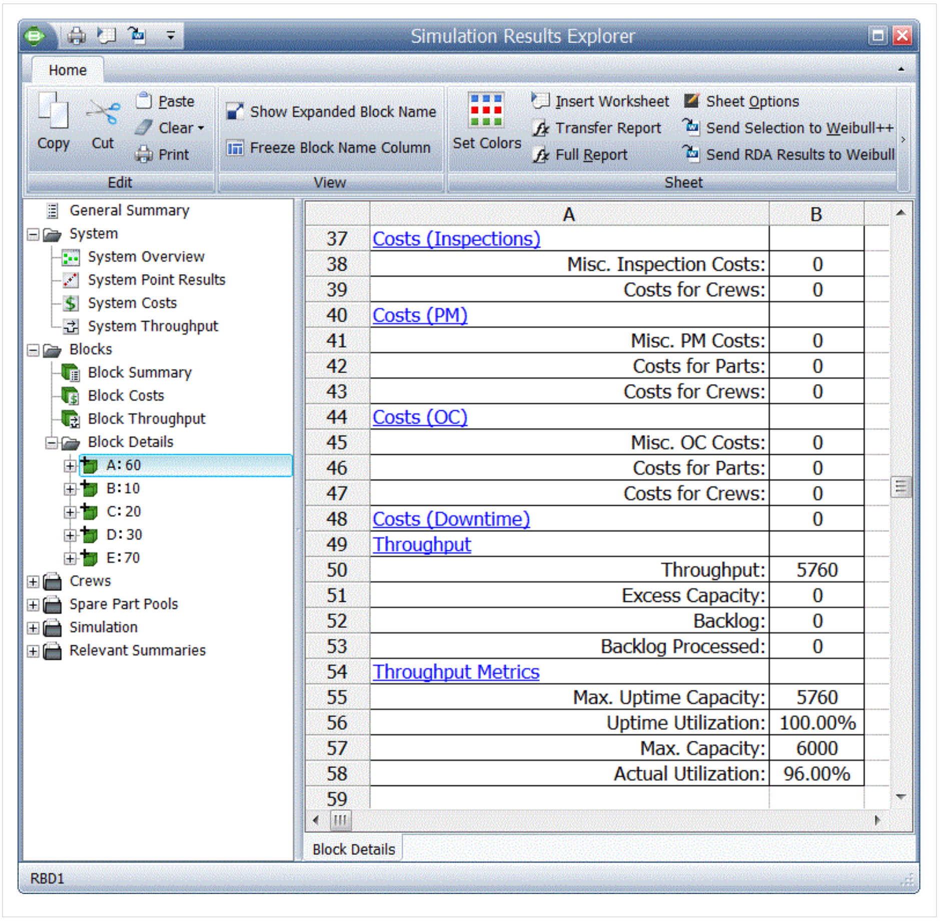
	A	B
1	System Throughput Summary	
2	Total Throughput:	5484
3		
4	Block Name	Uptime Utilization
5	A:60	100.00%
6	B:10	95.83%
7	C:20	95.83%
8	D:30	94.58%
9	E:70	81.61%

System level results present the total system throughput, which is 5484 items in this example. Additionally, the results include the uptime utilization of each component. The block level result summary, shown next, provides additional results for each item.



	A	B	C	D	E	F
1		Block Throughput Summary				
2	Block Name (Diagram)	Throughput	Excess Capacity	Backlog	Backlog Processed	Excess Backlog
3	A:60	5760	0	0	0	0
4	B:10	920	40	60	40	0
5	C:20	1840	80	96	80	0
6	D:30	2724	156	120	84	0
7	E:70	5484	1236	0	0	0

Finally, specific throughput results and metrics for each block are provided, as shown next.



	A	B
37	Costs (Inspections)	
38	Misc. Inspection Costs:	0
39	Costs for Crews:	0
40	Costs (PM)	
41	Misc. PM Costs:	0
42	Costs for Parts:	0
43	Costs for Crews:	0
44	Costs (OC)	
45	Misc. OC Costs:	0
46	Costs for Parts:	0
47	Costs for Crews:	0
48	Costs (Downtime)	0
49	Throughput	
50	Throughput:	5760
51	Excess Capacity:	0
52	Backlog:	0
53	Backlog Processed:	0
54	Throughput Metrics	
55	Max. Uptime Capacity:	5760
56	Uptime Utilization:	100.00%
57	Max. Capacity:	6000
58	Actual Utilization:	96.00%
59		

Life Cycle Cost Analysis

A life cycle cost analysis involves the analysis of the costs of a system or a component over its entire life span. Typical costs for a system may include:

- Acquisition costs (or design and development costs).
- Operating costs:
  - Cost of failures.
  - Cost of repairs.
  - Cost for spares.
  - Downtime costs.
  - Loss of production.
- Disposal costs.

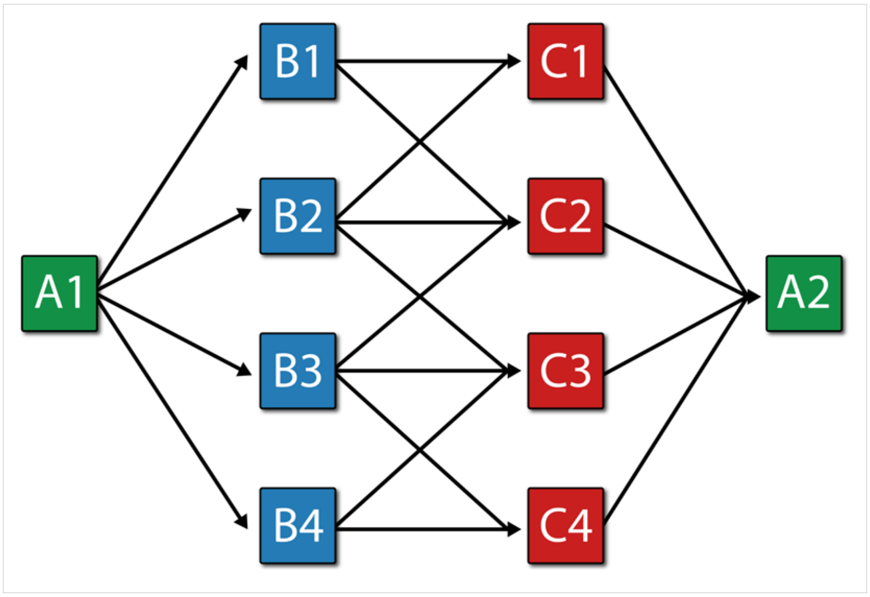
A complete life cycle cost (LCC) analysis may also include other costs, as well as other accounting/financial elements (such as discount rates, interest rates, depreciation, present value of money, etc.).

For the purpose of this reference, it is sufficient to say that if one has all the required cost values (inputs), then a complete LCC analysis can be performed easily in a spreadsheet, since it really involves summations of costs and perhaps some computations involving interest rates. With respect to the cost inputs for such an analysis, the costs involved are either deterministic (such as acquisition costs, disposal costs, etc.) or probabilistic (such as cost of failures, repairs, spares, downtime, etc.). Most of the probabilistic costs are directly related to the reliability and maintainability characteristics of the system.

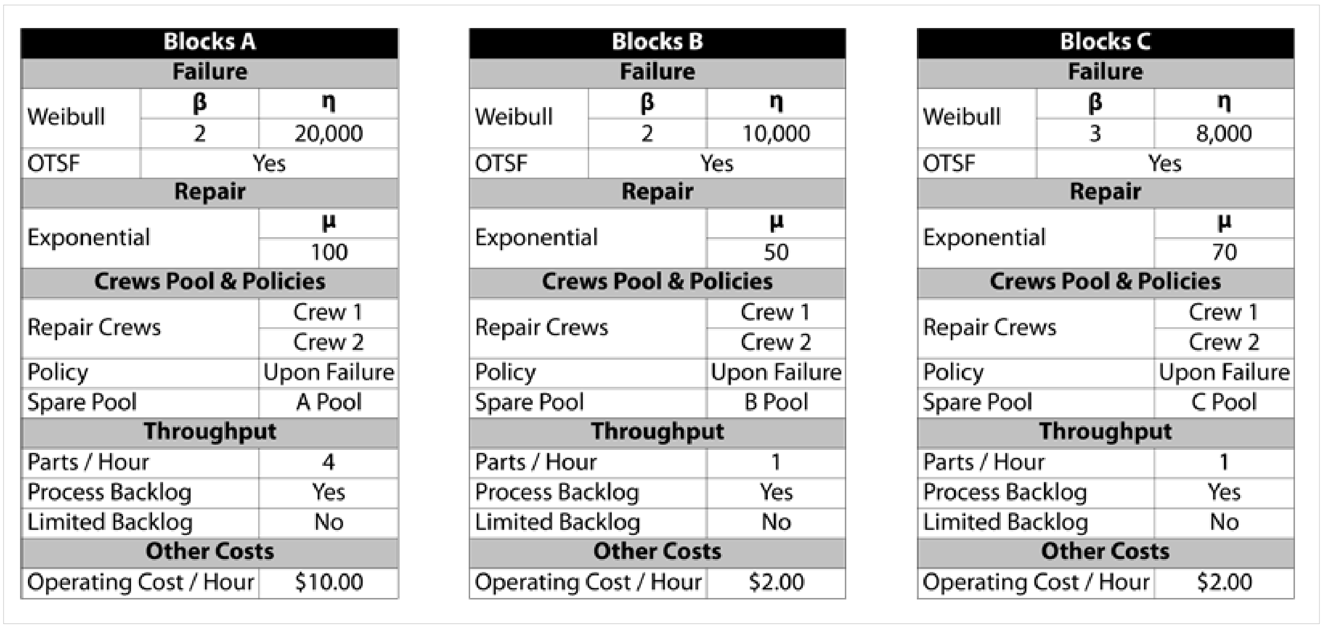
The estimations of the associated probabilistic costs is the challenging aspect of LCC analysis. In the following example, we will look at using some of the cost inputs associated with BlockSim to obtain such costs.

Example: Obtaining Costs for an LCC Analysis

Consider the manufacturing line (or system) shown next.



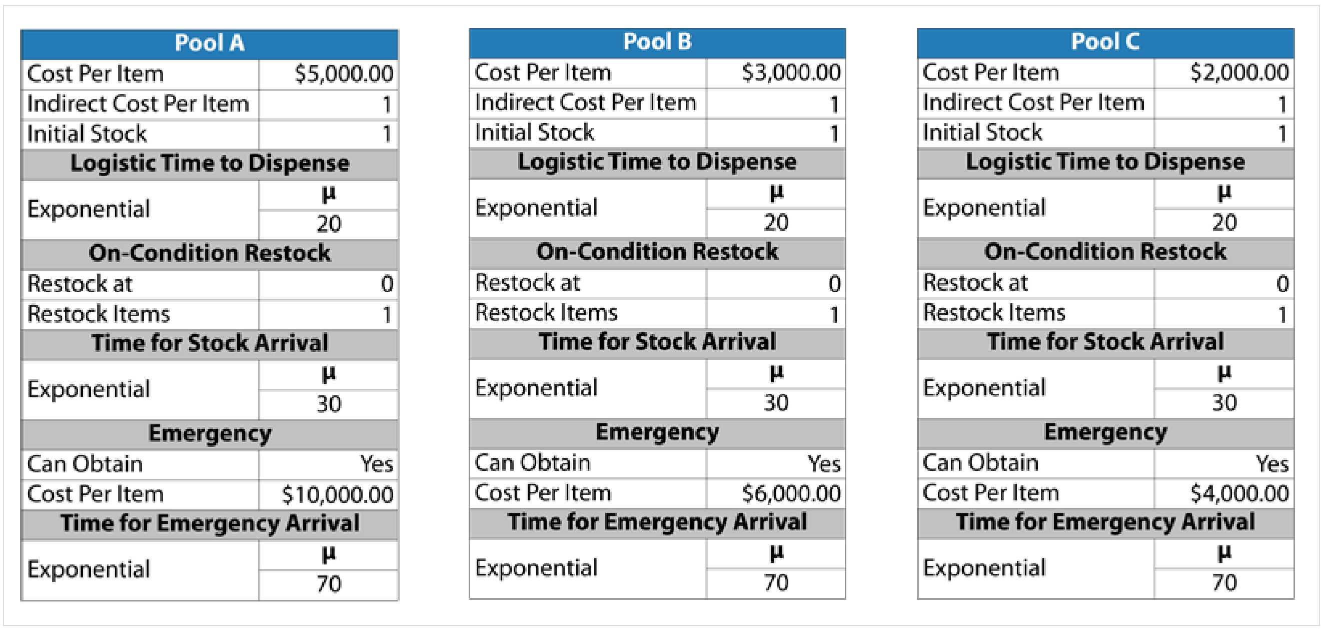
The block properties, pool properties and crew properties are given in the following tables. All blocks identified with the same letter have the same properties (i.e., Blocks A = A1 and A2; Blocks B = B1, B2, B3 and B4; and Blocks C = C1, C2, C3 and C4).



Blocks A		
Failure		
Weibull	β	η
	2	20,000
OTSF	Yes	
Repair		
Exponential		μ
		100
Crews Pool & Policies		
Repair Crews		Crew 1
		Crew 2
Policy		Upon Failure
Spare Pool		A Pool
Throughput		
Parts / Hour		4
Process Backlog		Yes
Limited Backlog		No
Other Costs		
Operating Cost / Hour		$10.00

Blocks B		
Failure		
Weibull	β	η
	2	10,000
OTSF	Yes	
Repair		
Exponential		μ
		50
Crews Pool & Policies		
Repair Crews		Crew 1
		Crew 2
Policy		Upon Failure
Spare Pool		B Pool
Throughput		
Parts / Hour		1
Process Backlog		Yes
Limited Backlog		No
Other Costs		
Operating Cost / Hour		$2.00

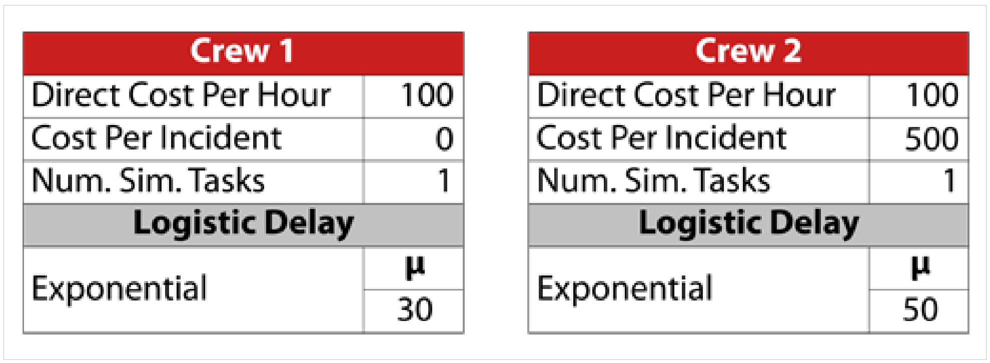
Blocks C		
Failure		
Weibull	β	η
	3	8,000
OTSF	Yes	
Repair		
Exponential		μ
		70
Crews Pool & Policies		
Repair Crews		Crew 1
		Crew 2
Policy		Upon Failure
Spare Pool		C Pool
Throughput		
Parts / Hour		1
Process Backlog		Yes
Limited Backlog		No
Other Costs		
Operating Cost / Hour		$2.00



Pool A	
Cost Per Item	$5,000.00
Indirect Cost Per Item	1
Initial Stock	1
Logistic Time to Dispense	
Exponential	µ
	20
On-Condition Restock	
Restock at	0
Restock Items	1
Time for Stock Arrival	
Exponential	µ
	30
Emergency	
Can Obtain	Yes
Cost Per Item	$10,000.00
Time for Emergency Arrival	
Exponential	µ
	70

Pool B	
Cost Per Item	$3,000.00
Indirect Cost Per Item	1
Initial Stock	1
Logistic Time to Dispense	
Exponential	µ
	20
On-Condition Restock	
Restock at	0
Restock Items	1
Time for Stock Arrival	
Exponential	µ
	30
Emergency	
Can Obtain	Yes
Cost Per Item	$6,000.00
Time for Emergency Arrival	
Exponential	µ
	70

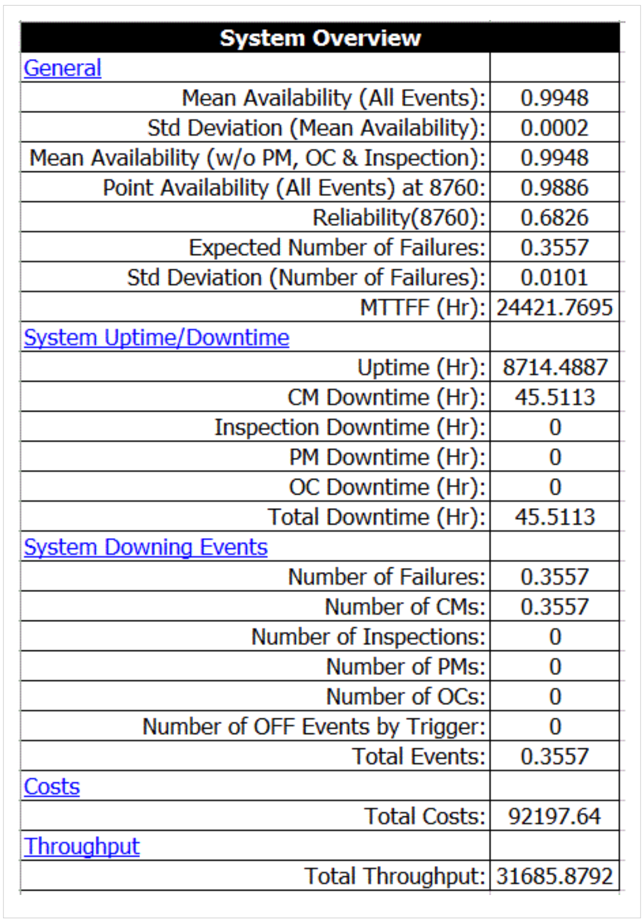
Pool C	
Cost Per Item	$2,000.00
Indirect Cost Per Item	1
Initial Stock	1
Logistic Time to Dispense	
Exponential	µ
	20
On-Condition Restock	
Restock at	0
Restock Items	1
Time for Stock Arrival	
Exponential	µ
	30
Emergency	
Can Obtain	Yes
Cost Per Item	$4,000.00
Time for Emergency Arrival	
Exponential	µ
	70



Crew 1	
Direct Cost Per Hour	100
Cost Per Incident	0
Num. Sim. Tasks	1
Logistic Delay	
Exponential	µ
	30

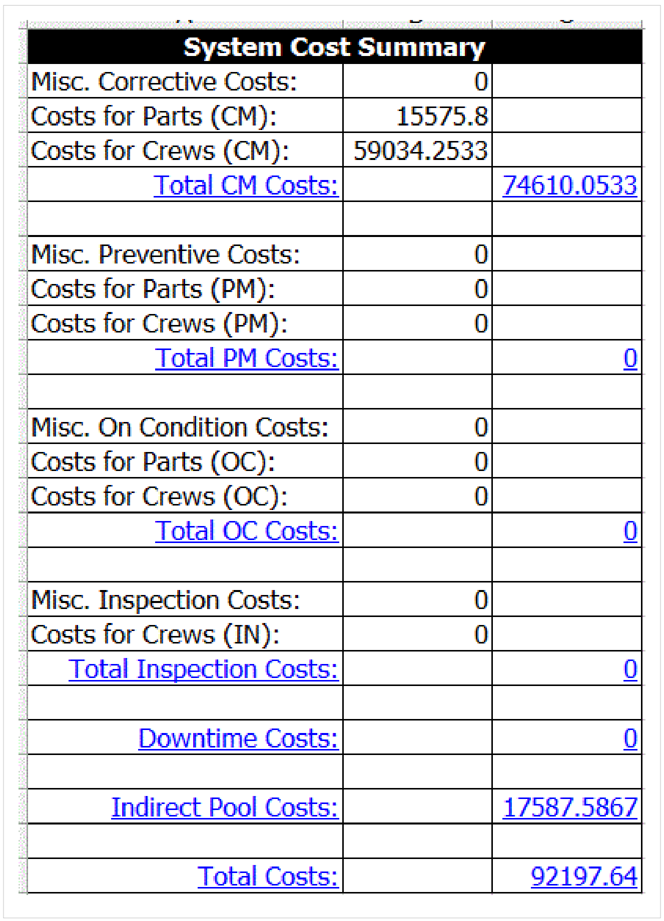
Crew 2	
Direct Cost Per Hour	100
Cost Per Incident	500
Num. Sim. Tasks	1
Logistic Delay	
Exponential	µ
	50

This system was analyzed in BlockSim for a period of operation of 8,760 hours, or one year. 10,000 simulations were performed. The system overview is shown next.



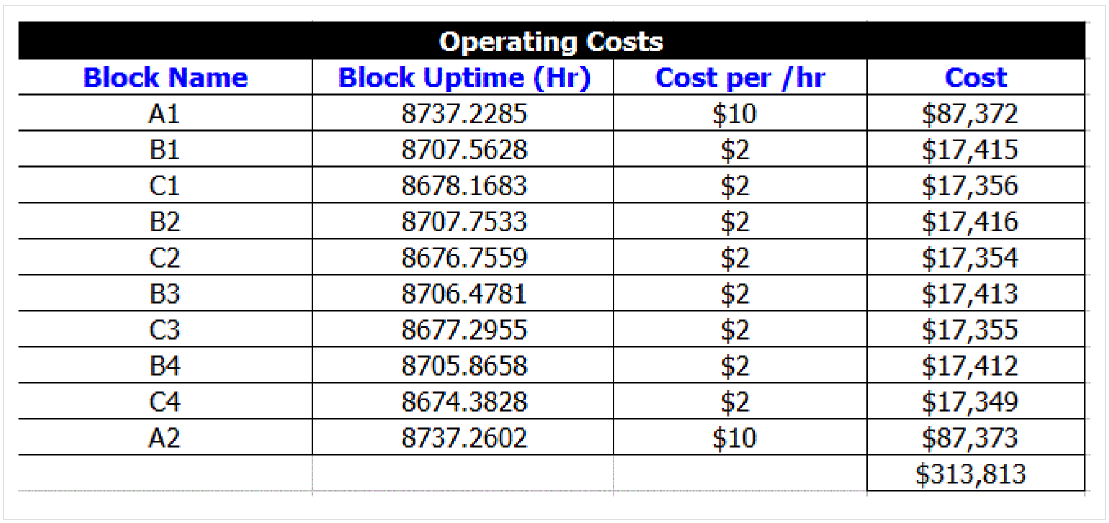
System Overview	
General	
Mean Availability (All Events):	0.9948
Std Deviation (Mean Availability):	0.0002
Mean Availability (w/o PM, OC & Inspection):	0.9948
Point Availability (All Events) at 8760:	0.9886
Reliability(8760):	0.6826
Expected Number of Failures:	0.3557
Std Deviation (Number of Failures):	0.0101
MTTFF (Hr):	24421.7695
System Uptime/Downtime	
Uptime (Hr):	8714.4887
CM Downtime (Hr):	45.5113
Inspection Downtime (Hr):	0
PM Downtime (Hr):	0
OC Downtime (Hr):	0
Total Downtime (Hr):	45.5113
System Downing Events	
Number of Failures:	0.3557
Number of CMs:	0.3557
Number of Inspections:	0
Number of PMs:	0
Number of OCs:	0
Number of OFF Events by Trigger:	0
Total Events:	0.3557
Costs	
Total Costs:	92197.64
Throughput	
Total Throughput:	31685.8792

Most of the variable costs of interest were obtained directly from BlockSim. The next figure shows the overall system costs.



System Cost Summary		
Misc. Corrective Costs:	0	
Costs for Parts (CM):	15575.8	
Costs for Crews (CM):	59034.2533	
Total CM Costs:		74610.0533
Misc. Preventive Costs:	0	
Costs for Parts (PM):	0	
Costs for Crews (PM):	0	
Total PM Costs:		0
Misc. On Condition Costs:	0	
Costs for Parts (OC):	0	
Costs for Crews (OC):	0	
Total OC Costs:		0
Misc. Inspection Costs:	0	
Costs for Crews (IN):	0	
Total Inspection Costs:		0
Downtime Costs:		0
Indirect Pool Costs:		17587.5867
Total Costs:		92197.64

From the summary, the total cost is $92,197.64. Note that an additional cost was defined in the problem statement that is not included in the summary. This cost, the operating cost per item per hour of operation, can be obtained by looking at the uptime of each block and then multiplying this by the cost per hour, as shown in the following table. Therefore, the total cost is $92,197 + 313,813 = \$406,010$.



Operating Costs			
Block Name	**Block Uptime (Hr)**	**Cost per /hr**	**Cost**
A1	8737.2285	$10	$87,372
B1	8707.5628	$2	$17,415
C1	8678.1683	$2	$17,356
B2	8707.7533	$2	$17,416
C2	8676.7559	$2	$17,354
B3	8706.4781	$2	$17,413
C3	8677.2955	$2	$17,355
B4	8705.8658	$2	$17,412
C4	8674.3828	$2	$17,349
A2	8737.2602	$10	$87,373
			$313,813

If we also assume a revenue of $100 per unit produced, then the total revenue is our throughput multiplied by the per unit revenue, or $31,685 \cdot \$100 = \$3,168,500$.

Chapter 9

Fault Tree Diagrams and System Analysis

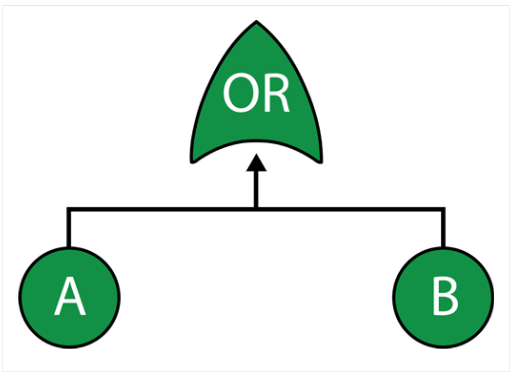
BlockSim allows system modeling using both reliability block diagrams (RBDs) and fault trees. This chapter introduces basic fault tree analysis and points out the similarities (and differences) between RBDs and fault tree diagrams. Principles, methods and concepts discussed in previous chapters are used.

Fault trees and reliability block diagrams are both symbolic analytical logic techniques that can be applied to analyze system reliability and related characteristics. Although the symbols and structures of the two diagram types differ, most of the logical constructs in a fault tree diagram (FTD) can also be modeled with a reliability block diagram (RBD). This chapter presents a brief introduction to fault tree analysis concepts and illustrates the similarities between fault tree diagrams and reliability block diagrams.

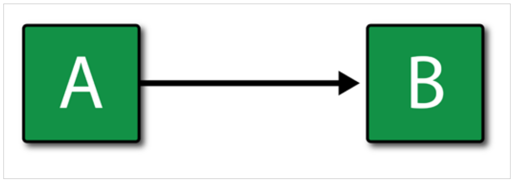
Fault Tree Analysis: Brief Introduction

Bell Telephone Laboratories developed the concept of fault tree analysis in 1962 for the U.S. Air Force for use with the Minuteman system. It was later adopted and extensively applied by the Boeing Company. A fault tree diagram follows a top-down structure and represents a graphical model of the pathways within a system that can lead to a foreseeable, undesirable loss event (or a failure). The pathways interconnect contributory events and conditions using standard logic symbols (AND, OR, etc.).

Fault tree diagrams consist of gates and events connected with lines. The AND and OR gates are the two most commonly used gates in a fault tree. To illustrate the use of these gates, consider two events (called "input events") that can lead to another event (called the "output event"). If the occurrence of either input event causes the output event to occur, then these input events are connected using an OR gate. Alternatively, if both input events must occur in order for the output event to occur, then they are connected by an AND gate. The following figure shows a simple fault tree diagram in which either *A* or *B* must occur in order for the output event to occur. In this diagram, the two events are connected to an OR gate. If the output event is system failure and the two input events are component failures, then this fault tree indicates that the failure of *A* or *B* causes the system to fail.



The RBD equivalent for this configuration is a simple series system with two blocks, *A* and *B*, as shown next.

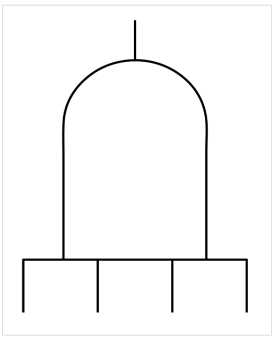


Basic Gates

Gates are the logic symbols that interconnect contributory events and conditions in a fault tree diagram. The AND and OR gates described above, as well as a Voting OR gate in which the output event occurs if a certain number of the input events occur (i.e., *k*-out-of-*n* redundancy), are the most basic types of gates in classical fault tree analysis. These gates are explicitly provided for in BlockSim and are described in this section along with their BlockSim implementations. Additional gates are introduced in the following sections.

A fault tree diagram is always drawn in a top-down manner with lowest item being a basic event block. Classical fault tree gates have no properties (i.e., they cannot fail).

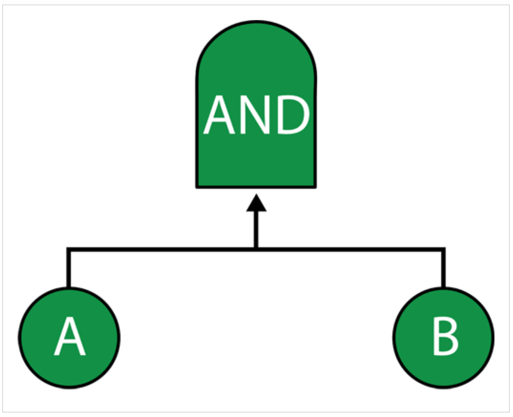
AND Gate

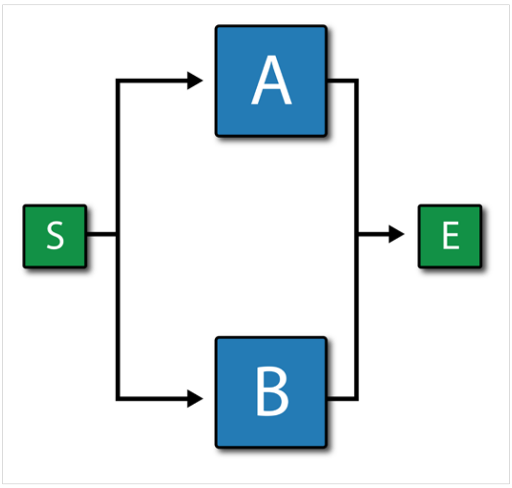


In an AND gate, the output event occurs if all input events occur. In system reliability terms, this implies that all components must fail (input) in order for the system to fail (output). When using RBDs, the equivalent is a simple parallel configuration.

Example

Consider a system with two components, *A* and *B*. The system fails if both *A* and *B* fail. Draw the fault tree and reliability block diagram for the system. The next two figures show both the FTD and RBD representations.

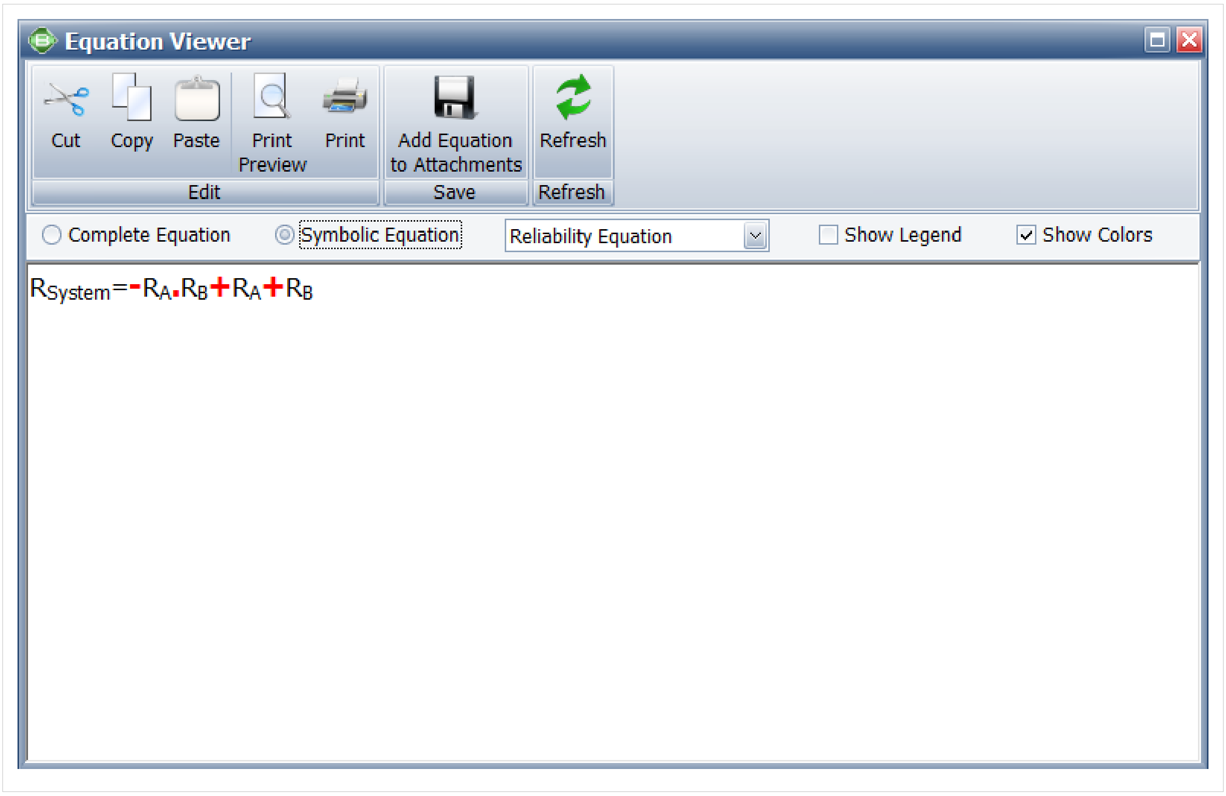




The reliability equation for either configuration is:

$$R_{System} = R_A + R_B - R_A \cdot R_B$$

The figure below shows the analytic equation from BlockSim.

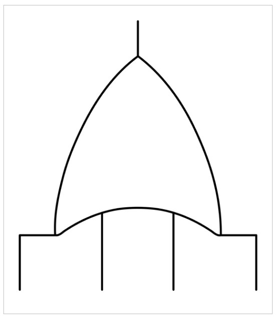


( ) Complete Equation [✓ Symbolic Equation] Reliability Equation ☐ Show Legend ☑ Show Colors

$R_{System} = -R_A \cdot R_B + R_A + R_B$

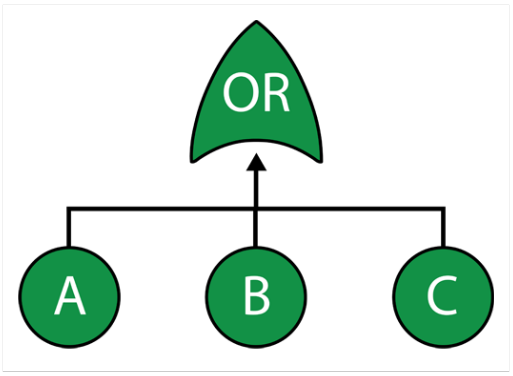
OR Gate

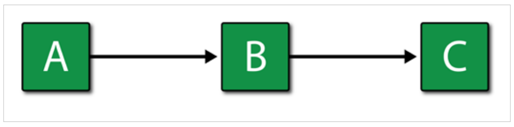
In an OR gate, the output event occurs if at least one of the input events occurs. In system reliability terms, this implies that if any component fails (input) then the system will fail (output). When using RBDs, the equivalent is a series configuration.



Example

Consider a system with three components, *A*, *B* and *C*. The system fails if *A*, *B* or *C* fails. Draw the fault tree and reliability block diagram for the system. The next two figures show both the FTD and RBD representations.

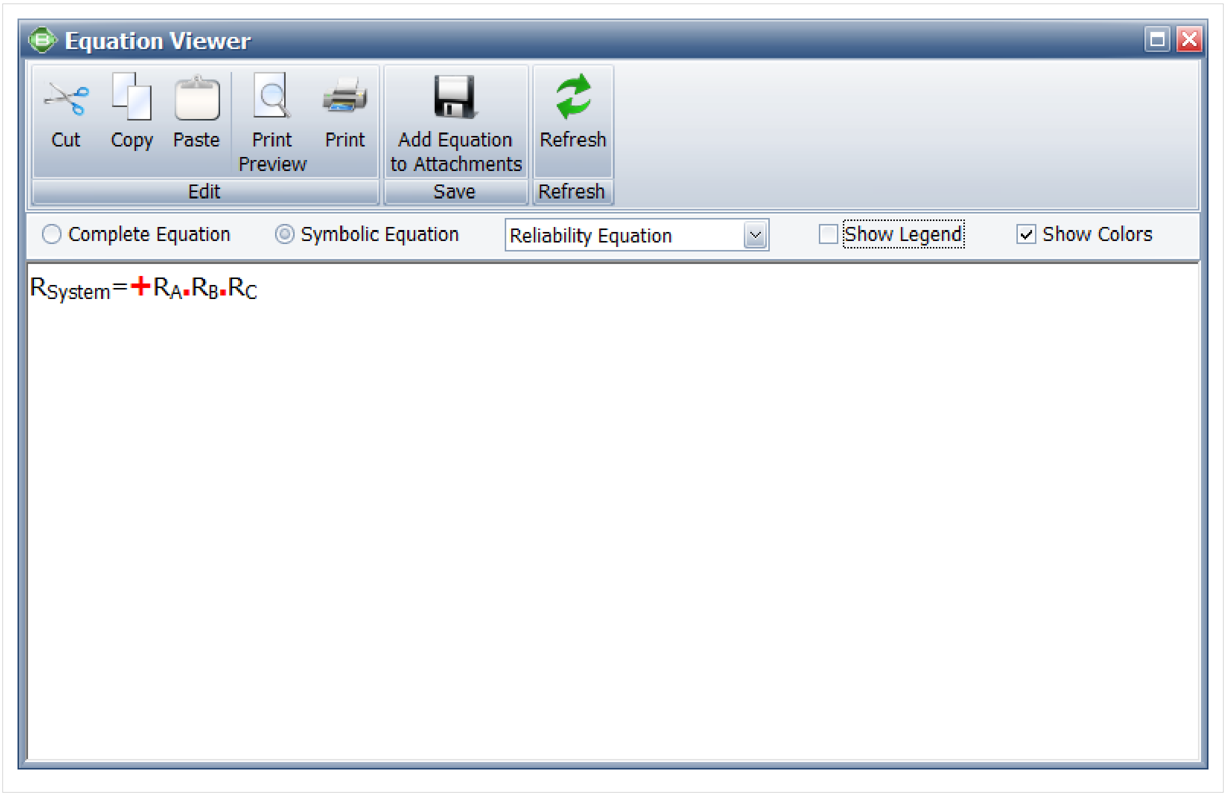




The reliability equation for either configuration is:

$$R_{System} = R_A \cdot R_B \cdot R_C$$

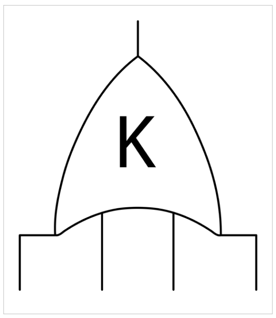
The figure below shows the analytic equation from BlockSim.



( ) Complete Equation [✓ Symbolic Equation] Reliability Equation ☐ Show Legend ☑ Show Colors

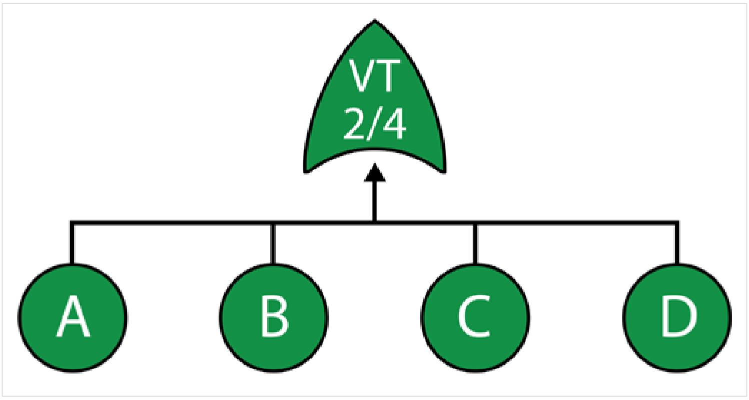
$R_{System}=+R_A \cdot R_B \cdot R_C$

Voting OR Gate

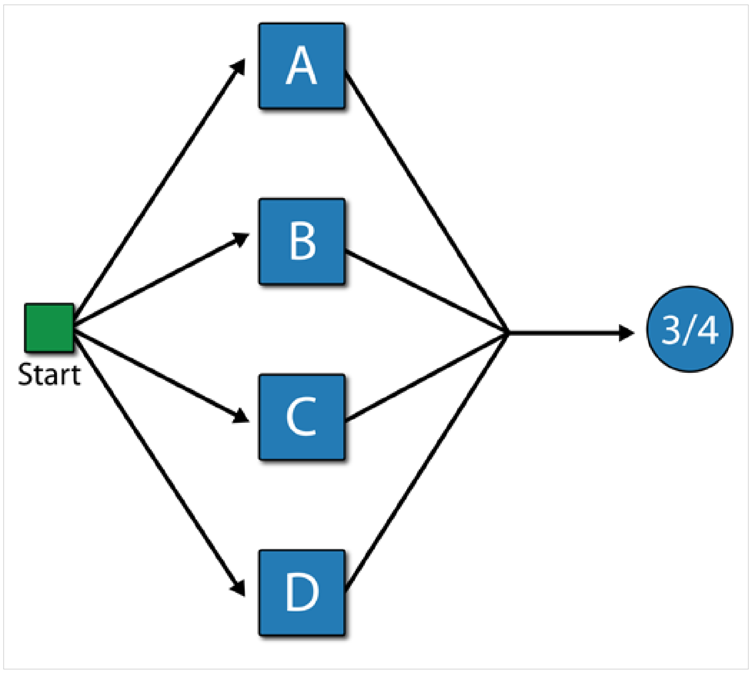


In a Voting OR gate, the output event occurs if *k* or more of the input events occur. In system reliability terms, this implies that if any *k*-out-of-*n* components fail (input) then the system will fail (output).

The equivalent RBD construct is a node and is similar to a *k*-out-of-*n* parallel configuration with a distinct difference, as discussed next. To illustrate this difference, consider a fault tree diagram with a 2-out-of-4 Voting OR gate, as shown in the following figure.



In this diagram, the system will fail if any two of the blocks below fail. Equivalently, this can be represented by the RBD shown in the next figure using a 3-out-of-4 node.



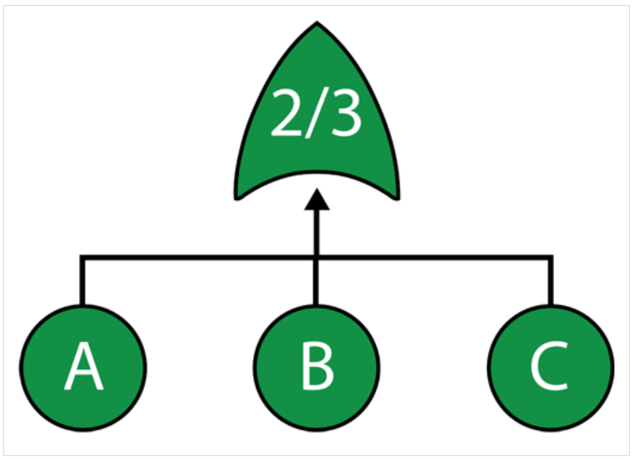
In this configuration, the system will not fail if three out of four components are operating, but will fail if more than one fails. In other words, the fault tree considers *k*-out-of-*n* failures for the system failure while the RBD considers *k*-out-of-*n* successes for system success.

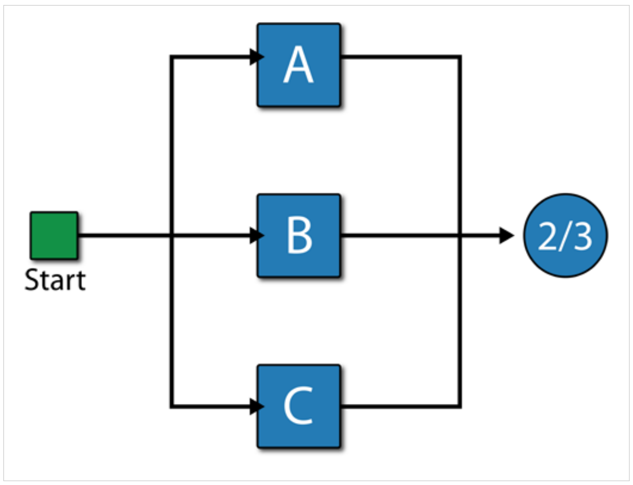
Increasing the Flexibility

Classical Voting OR gates have no properties and cannot fail or be repaired (i.e., they cannot be an event themselves). In BlockSim, Voting OR gates behave like nodes in an RBD; thus, they can also fail and be repaired just like any other event. By default, when a Voting OR gate is inserted into a fault tree diagram within BlockSim, the gate is set so that it cannot fail (classical definition). However, this property can be modified to allow for additional flexibility.

Example

Consider a system with three components, *A*, *B* and *C*. The system fails if any two components fail. Draw the fault tree and reliability block diagram for the system. The next two figures show both the FTD and RBD representations.





The reliability equation for either configuration is:

$$R_{System} = -2 \cdot R_A \cdot R_B \cdot R_C + R_A \cdot R_B + R_A \cdot R_C + R_B \cdot R_C$$

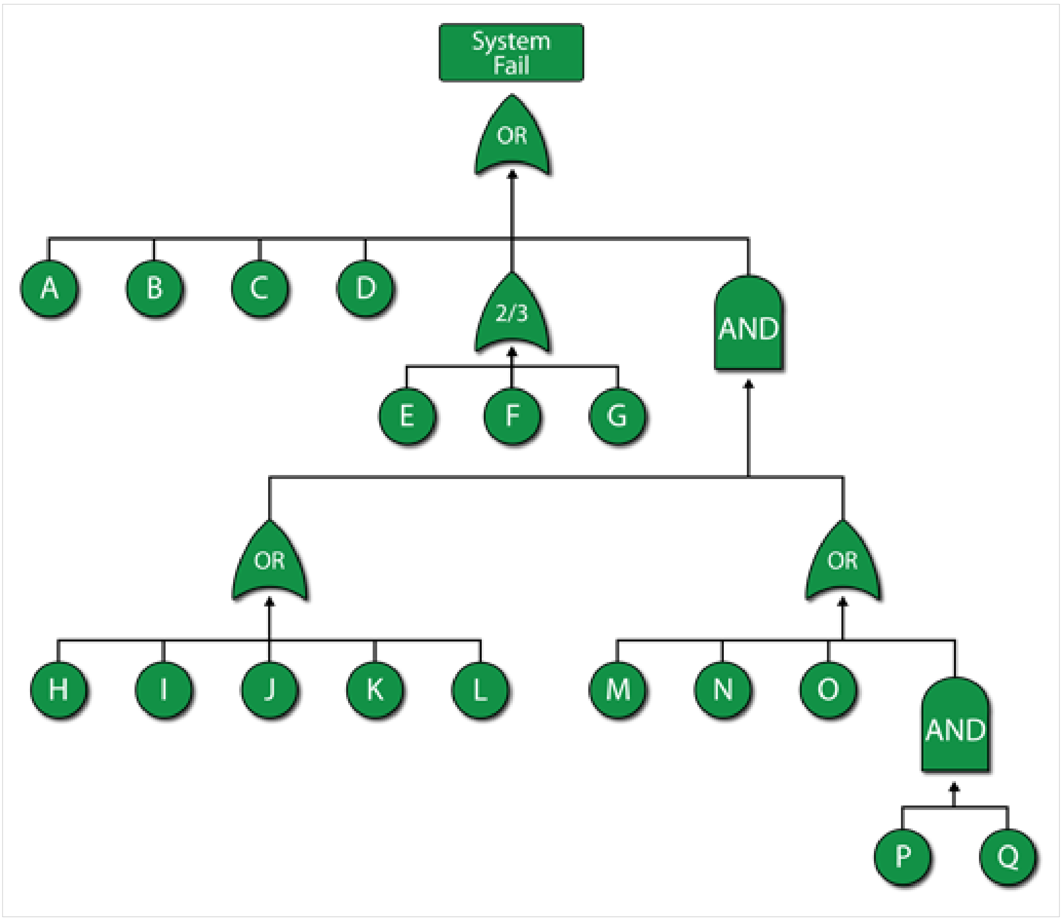
Equation above assumes a classical Voting OR gate (i.e., the voting gate itself cannot fail). If the gate can fail then the equation is modified as follows:

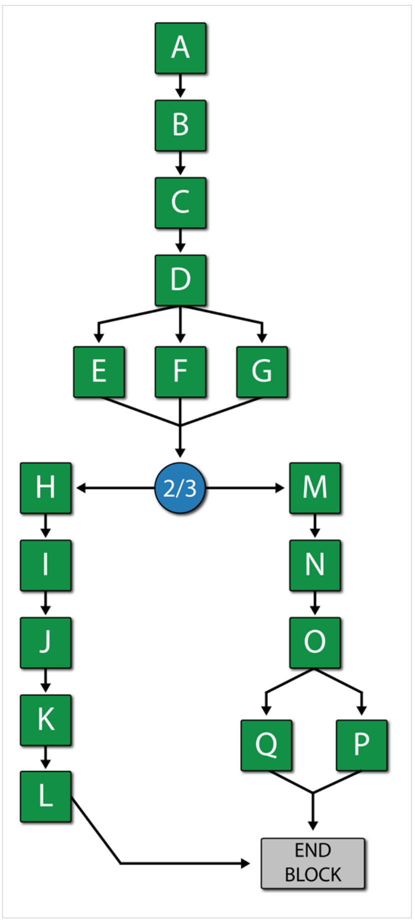
$$R_{System} = R_{Voting}(-2 \cdot R_A \cdot R_B \cdot R_C + R_A \cdot R_B + R_A \cdot R_C + R_B \cdot R_C)$$

Note that while both the gate and the node are 2-out-of-3, they represent different circumstances. The Voting OR gate in the fault tree indicates that if two components fail then the system will fail; while the node in the reliability block diagram indicates that if at least two components succeed then the system will succeed.

Combining Basic Gates

As in reliability block diagrams where different configuration types can be combined in the same diagram, fault tree analysis gates can also be combined to create more complex representations. As an example, consider the fault tree diagram shown in the figures below.

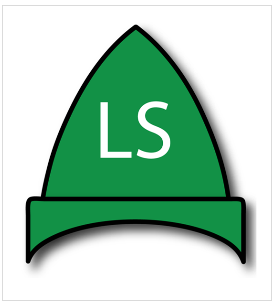




New BlockSim Gates

In addition to the gates defined above, other gates exist in classical FTA. These additional gates (e.g., Sequence Enforcing, Priority AND, etc.) are usually used to describe more complex redundancy configurations and are described in later sections. First, we will introduce two new advanced gates that can be used to append to and/or replace classical fault tree gates. These two new gates are the Load Sharing and Standby gates. Classical fault trees (or any other fault tree standard to our knowledge) do not allow for load sharing redundancy (or event dependency). To overcome this limitation, and to provide fault trees with the same flexibility as BlockSim's RBDs, we will define a Load Sharing gate in this section. Additionally, traditional fault trees do not provide the full capability to model standby redundancy configurations (including the quiescent failure distribution), although basic standby can be represented in traditional fault tree diagrams using a Priority AND gate or a Sequence Enforcing gate, discussed in later sections.

Load Sharing Gate



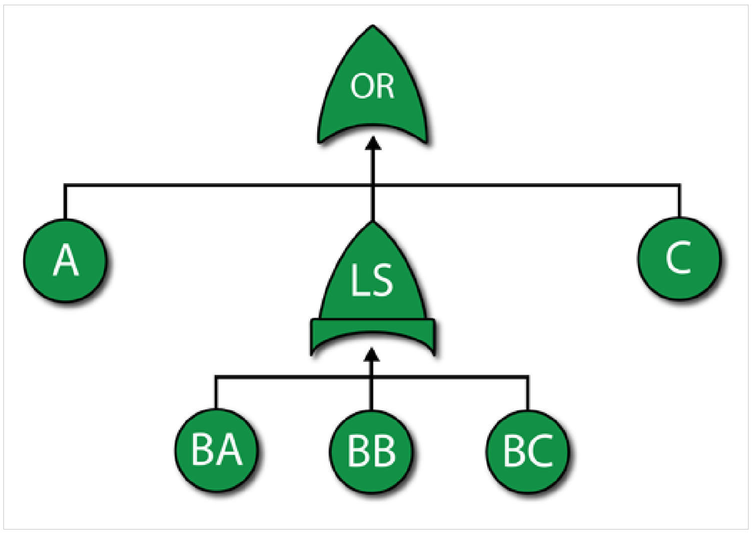
A Load Sharing gate behaves just like BlockSim's Load Sharing containers for RBDs. Load Sharing containers were discussed in Time-Dependent System Reliability (Analytical) and RBDs and Analytical System Reliability. Events leading into a Load Sharing gate have distributions and life-stress relationships, just like contained blocks. Furthermore, the gate defines the load and the number required to cause the output event (i.e., the Load Sharing gate is defined with a *k*-out-of-*n* vote). In BlockSim, no additional gates are allowed below a Load Sharing gate.

Example

A component has five possible failure modes, A, B_A, B_B, $B_C$and C, and the Bmodes are interdependent. The system will fail if mode Aoccurs, mode Coccurs or two out of the three Bmodes occur. Modes $A$and Chave a Weibull distribution with $\beta = 2$and $\eta = 10,000$and $15,000$respectively. Events B_A, $B_B$and $B_C$have an exponential distribution with a mean of $10,000$hours. If any Bevent occurs (i.e., B_A, $B_B$or B_C), the remaining Bevents are more likely to occur. Specifically, the mean times of the remaining Bevents are halved. Determine the reliability at 1,000 hours for this component.

Solution

The first step is to create the fault tree as shown in the figure below. Note that both an OR gate and a Load Sharing gate are used.



The next step is to define the properties for each event block and the Load Sharing gate. Setting the failure distributions for modes $A$and $C$is simple.

The more difficult part is setting the properties of the Load Sharing gate (which are the same as an RBD container) and the dependent load sharing events (which are the same as the contained blocks in an RBD). Based on the problem statement, the Bmodes are in a 2-out-of-3 load sharing redundancy. When all three are working (i.e., when no Bmode has occurred), each block has an exponential distribution with $\mu = 10,000$. If one Bmode occurs, then the two surviving units have an exponential distribution with $\mu = 5,000$.

Assume an inverse power life-stress relationship for the components. Then:

$$\mu_1 = \frac{1}{KV_1^n}$$

$$\mu_2 = \frac{1}{KV_2^n}$$

Substituting $\mu_1 = 10,000$and $V_1 = 1$in $\mu_1 = \dfrac{1}{KV_1^n}$and casting it in terms of Kyields:

$$10,000 = \frac{1}{K}$$

$$K = \frac{1}{10,000} = 0.0001$$

Substituting $\mu_2 = 5,000$, $V_2 = 1.5$(because if one fails, then each survivor takes on an additional 0.5 units of load) and $10,000 = \frac{1}{K}$for Kin $\mu_2 = \frac{1}{KV_2^n}$yields:

$$5,000 = \frac{1}{0.0001 \cdot (1.5)^n}$$

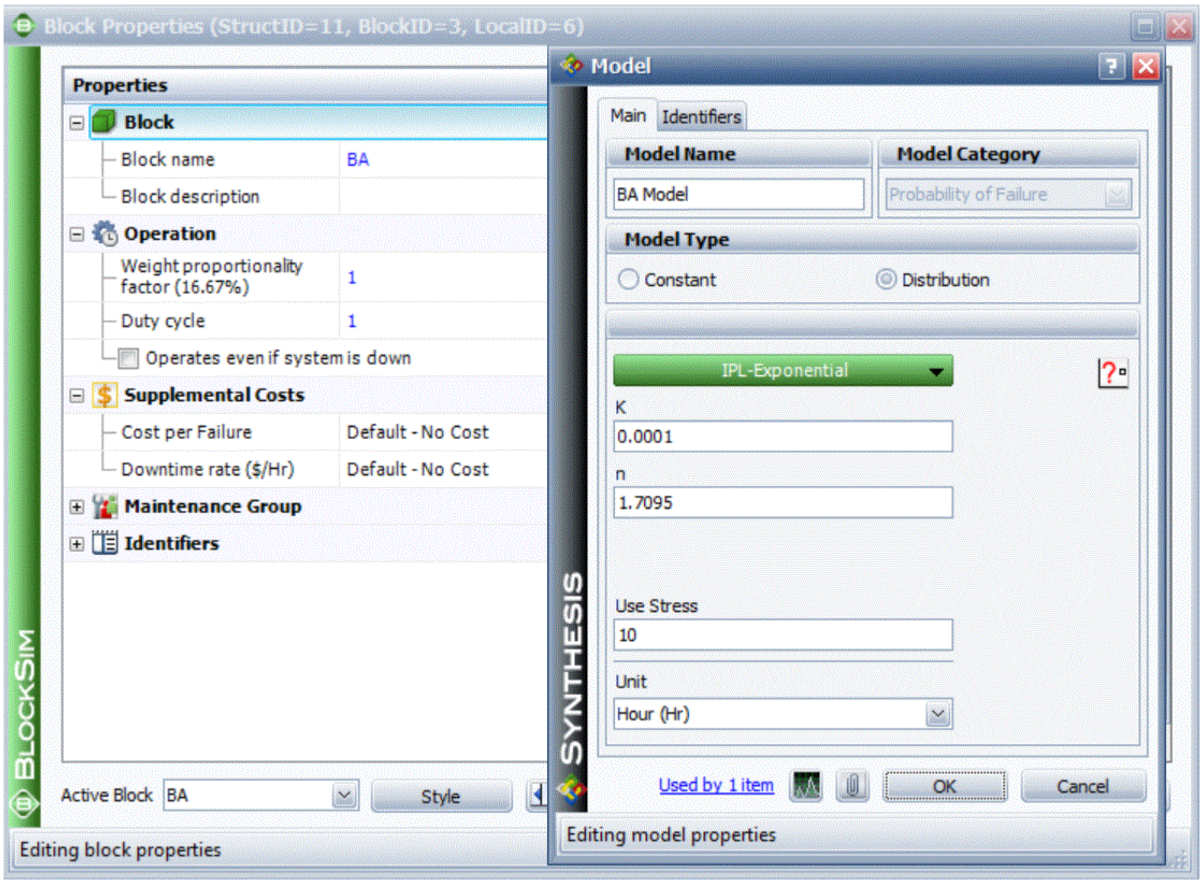
$$0.5 = (1.5)^{-n}$$

$$\ln(0.5) = -n\ln(1.5)$$

$$n = 1.7095$$

This also could have been computed in ReliaSoft's ALTA software or with the Load & Life Parameter Experimenter in BlockSim. This was done in Time-Dependent System Reliability (Analytical) .

At this point, the parameters for the load sharing units have been computed and can be set, as shown in the following figure. (Note: when define the IPL-Exponential model, we just need to specify the value for K and n, the value for Use Stress is not a issue here, leave it as default number 10 or any number will be good.)



Block Properties (StructID=11, BlockID=3, LocalID=6)

Properties	
Block	
Block name	BA
Block description	
Operation	
Weight proportionality factor (16.67%)	1
Duty cycle	1
☐ Operates even if system is down	
Supplemental Costs	
Cost per Failure	Default - No Cost
Downtime rate ($/Hr)	Default - No Cost
Maintenance Group	
Identifiers	

Active Block: BA

Editing block properties

Model

Model Name: BA Model
Model Category: Probability of Failure
Model Type: ◯ Constant [✓ Distribution]
IPL-Exponential
K: 0.0001
n: 1.7095
Use Stress: 10
Unit: Hour (Hr)

Editing model properties

The next step is to set the weight proportionality factor. This factor defines the portion of the load that the particular item carries while operating, as well as the load that shifts to the remaining units upon failure of the item. To illustrate, assume three units (1, 2 and 3) are in a load sharing redundancy, represented in the fault tree diagram by a Load Sharing gate, with weight proportionality factors of 1, 2 and 3 respectively (and a 3-out-of-3 requirement).

- Unit 1 carries $\left(\frac{1}{1+2+3}\right) = 0.166$or 16.6% of the total load.
- Unit 2 carries $\left(\frac{2}{1+2+3}\right) = 0.333$or 33.3% of the total load.

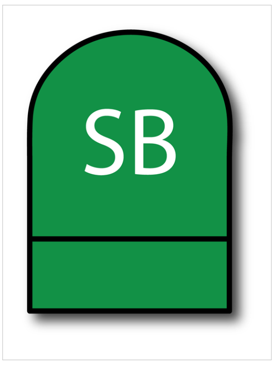
- Unit 3 carries $\left(\frac{3}{1+2+3}\right) = 0.50$or 50% of the total load.

The actual load on each unit then becomes the product of the entire load defined for the gate multiplied by the portion carried by that unit. For example, if the load is 100 lbs, then the portion assigned to Unit 1 will be $100 \cdot 0.166 = 16.6$lbs.

In the current example, all units share the same load; thus, they have equal weight proportionality factors. Because these factors are relative, if the same number is used for all three items then the results will be the same. For simplicity, we will set the factor equal to 1 for each item.

Once the properties have been specified in BlockSim, the reliability at 1000 hours can be determined. From the Analytical QCP, this is found to be 93.87%.

Standby Gate



A Standby gate behaves just like a standby container in BlockSim's RBDs. Standby containers were discussed in Time-Dependent System Reliability (Analytical) and RBDs and Analytical System Reliability. Events leading into a Standby gate have active and quiescent failure distributions, just like contained blocks. Furthermore, the gate acts as the switch, can fail and can also define the number of active blocks whose failure would cause system failure (i.e., the Active Vote Number required). In BlockSim, no additional gates are allowed below a Standby gate.

Example

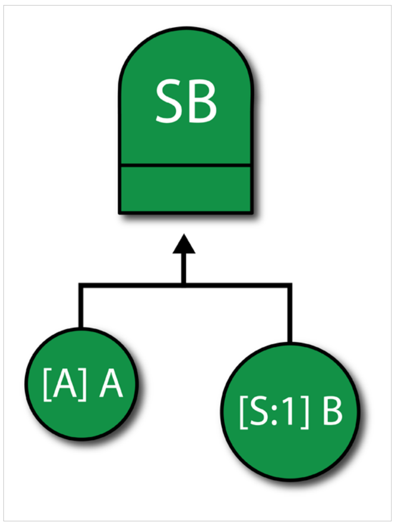
Consider a system with two units, *A* and *B*, in a standby configuration. Unit *A* is active and unit *B* is in a "warm" standby configuration. Furthermore, assume perfect switching (i.e., the switch cannot fail and the switch occurs instantly). Units *A* and *B* have the following failure properties:

- Block A(Active):
  - Failure Distribution: Weibull; $\beta = 1.5$; $\eta = 1,000$hours.
- Block B(Standby):
  - Energized failure distribution: Weibull; $\beta = 1.5$; $\eta = 1,000$hours.
  - Quiescent failure distribution: Weibull; $\beta = 1.5$; $\eta = 2,000$hours.

Determine the reliability of the system for 500 hours.

Solution

The fault tree diagram for this configuration is shown next and $R(t = 500) = 94.26\%$.



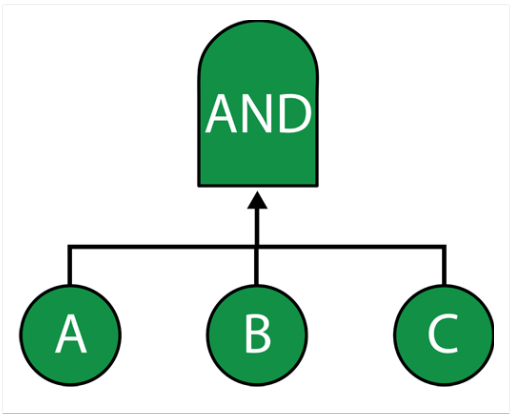
Additional Classical Gates and Their Equivalents in BlockSim

Sequence Enforcing Gate

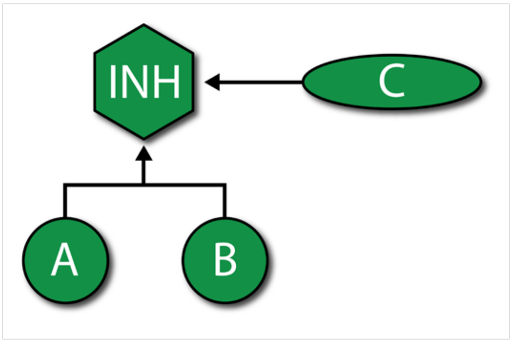
Various graphical symbols have been used to represent a Sequence Enforcing gate. It is a variation of an AND gate in which each item must happen in sequence. In other words, events are constrained to occur in a specific sequence and the output event occurs if all input events occur in that specified sequence. This is identical to a cold standby redundant configuration (i.e., k units in standby with no quiescent failure distribution and no switch failure probability). BlockSim does not explicitly provide a Sequence Enforcing gate; however, it can be easily modeled using the more advanced Standby gate, described previously.

Inhibit Gate

In an Inhibit gate, the output event occurs if all input events occur and an additional conditional event occurs. It is an AND gate with an additional event. In reality, an Inhibit gate provides no additional modeling capabilities but is used to illustrate the fact that an additional event must also occur. As an example, consider the case where events *A* and *B* must occur as well as a third event *C* (the so-called conditional event) in order for the system to fail. One can represent this in a fault tree by using an AND gate with three events, *A*, *B* and *C*, as shown next.

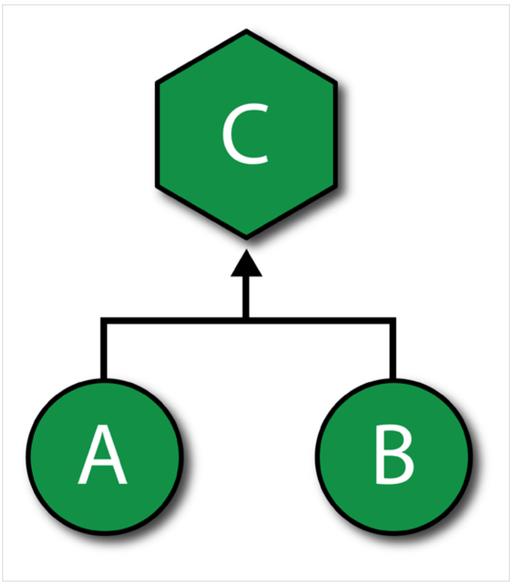


Classical fault tree diagrams have the conditional event drawn to the side and the gate drawn as a hexagon, as shown next.

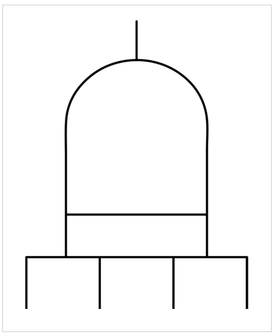


It should be noted that both representations are equivalent from an analysis standpoint.

BlockSim explicitly provides an Inhibit gate. This gate functions just like an AND gate with the exception that failure/repair characteristics can be assigned to the gate itself. This allows the construction shown above (if the gate itself is set to not fail). Additionally, one could encapsulate event *C* inside the gate (since the gate can have properties), as shown next. Note that all three figures can be represented using a single RBD with events *A*, *B* and *C* in parallel.

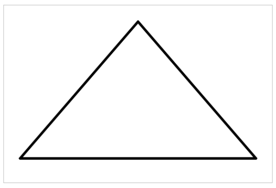


Priority AND Gate

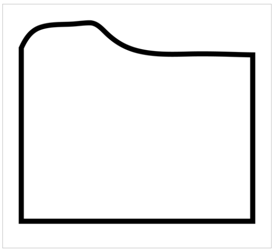


With a Priority AND gate, the output event occurs if all input events occur in a specific sequence. This is an AND gate that requires that all events occur in a specific sequence. At first, this may seem identical to the Sequence Enforcing gate discussed earlier. However, it differs from this gate in the fact that events can occur out of sequence (i.e., are not constrained to occur in a specific sequence) but the output event only occurs if the sequence is followed. To better illustrate this, consider the case of two motors in standby configuration with motor Abeing the primary motor and motor *B* in standby. If motor *A* fails, then the switch (which can also fail) activates motor *B*. Then the system will fail if motor *A* fails and the switch fails to switch, or if the switch succeeds but motor *B* fails subsequent to the switching action. In this scenario, the events must occur in the order noted; however, it is possible for the switch or motor *B* to fail (in a quiescent mode) without causing a system failure, if *A* never fails. BlockSim does not explicitly provide a Priority AND gate. However, like the Sequence Enforcing gate, it can be easily modeled using the more advanced Standby gate.

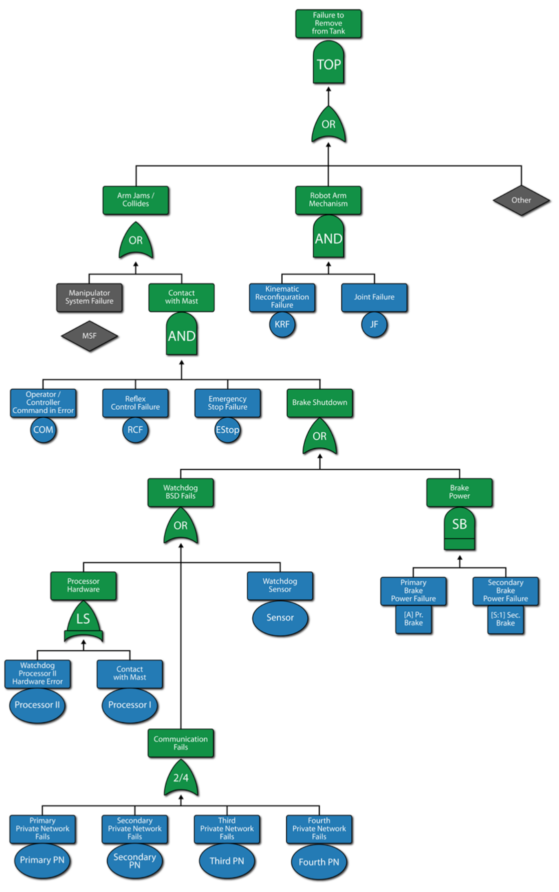
Transfer Gate



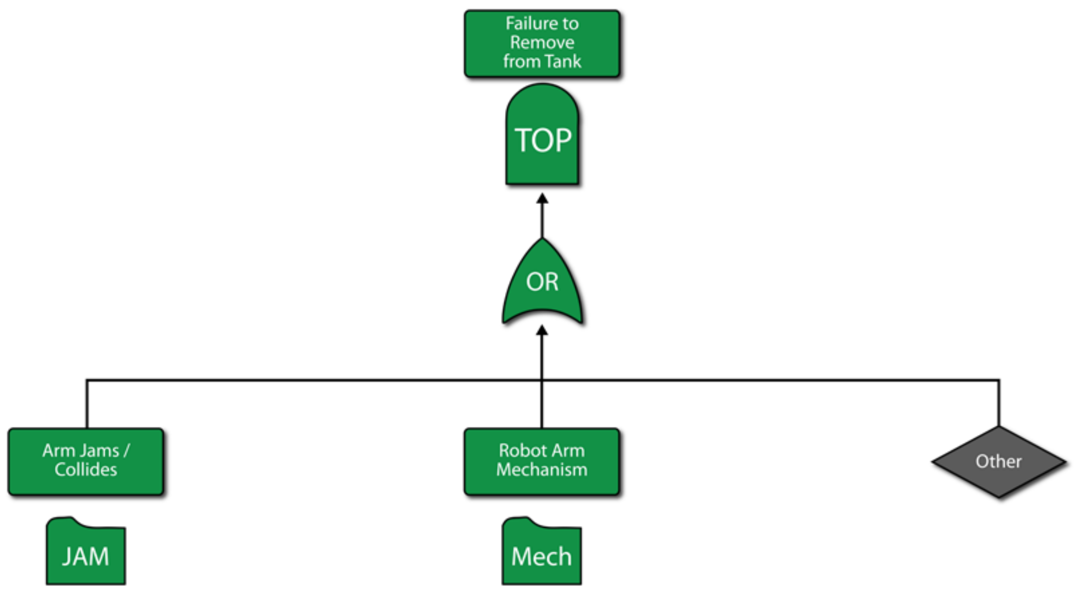
Transfer in/out gates are used to indicate a transfer/continuation of one fault tree to another. In classical fault trees, the Transfer gate is generally used to signify the continuation of a tree on a separate sheet. This is the same as a subdiagram block in an RBD. BlockSim does not explicitly provide a Transfer gate. However, it does allow for subdiagrams (or sub-trees), which provide for greater flexibility. Additionally, a subdiagram in a BlockSim fault tree can be an RBD and vice versa. BlockSim uses the more intuitive folder symbol to represent subdiagrams.



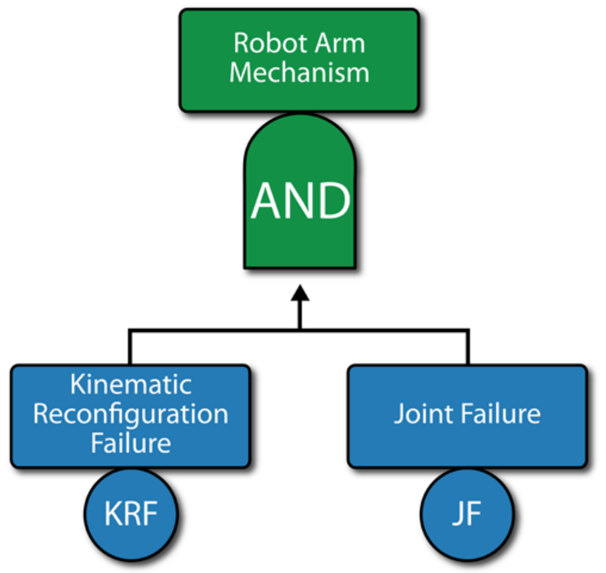
As an example, consider the fault tree of the robot manipulator shown in the first figure ("A") below. The second figure ("B") illustrates the same fault tree with the use of subdiagrams (Transfer gates). The referenced subdiagrams are shown in subsequent figures. Note that this is using multiple levels of indenture (i.e., the subdiagram has subdiagrams and so forth).



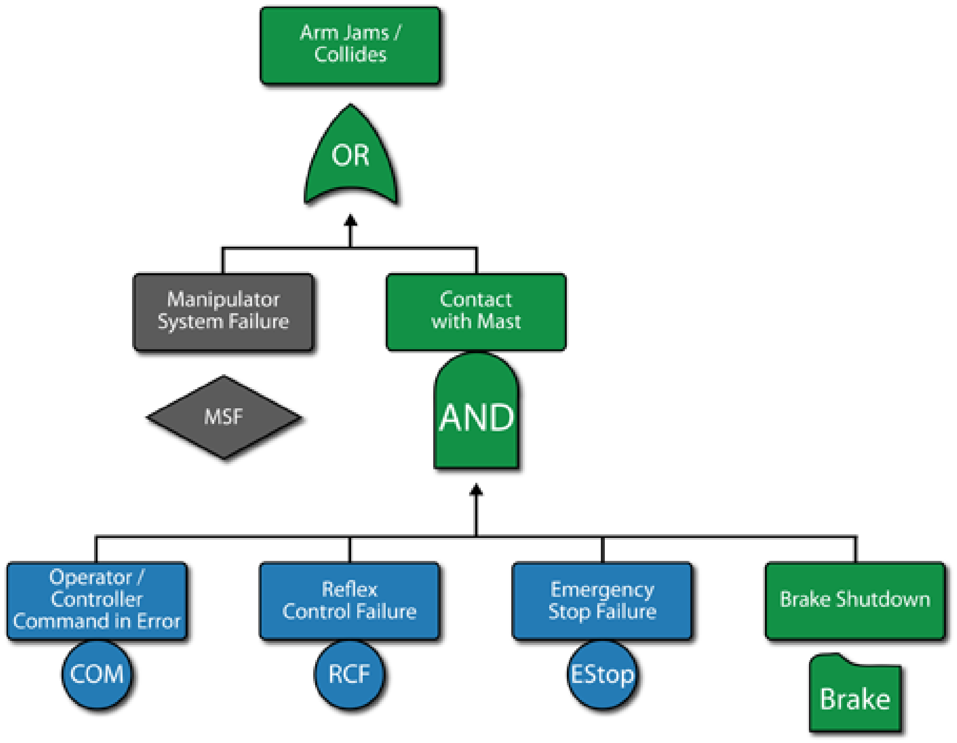
A: A sample fault tree for a robot manipulator, showing all items in a single tree.



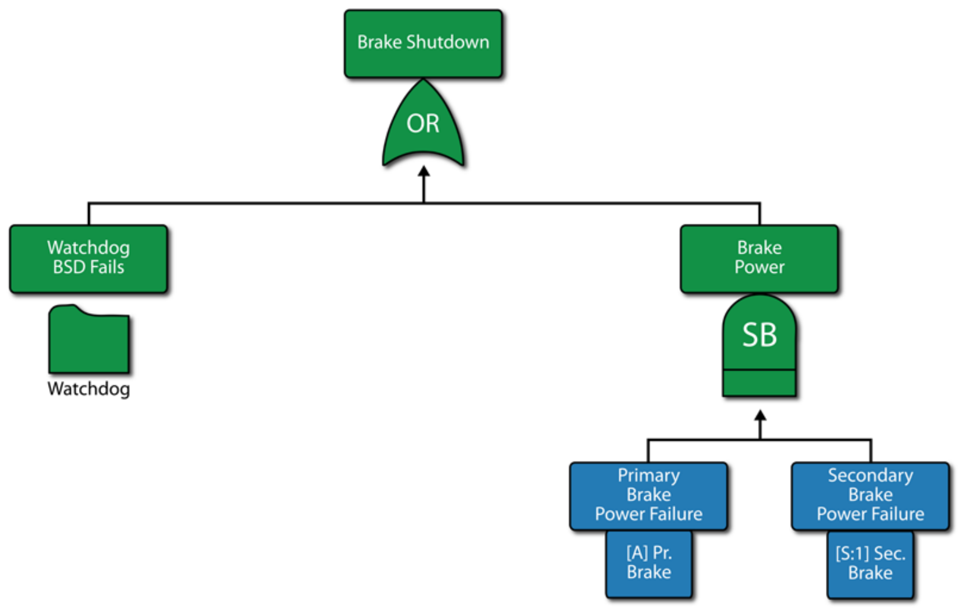
B: The fault tree of Figure A using subdiagrams (Transfer gates). The subdiagrams are shown in Figures "C" and "D".



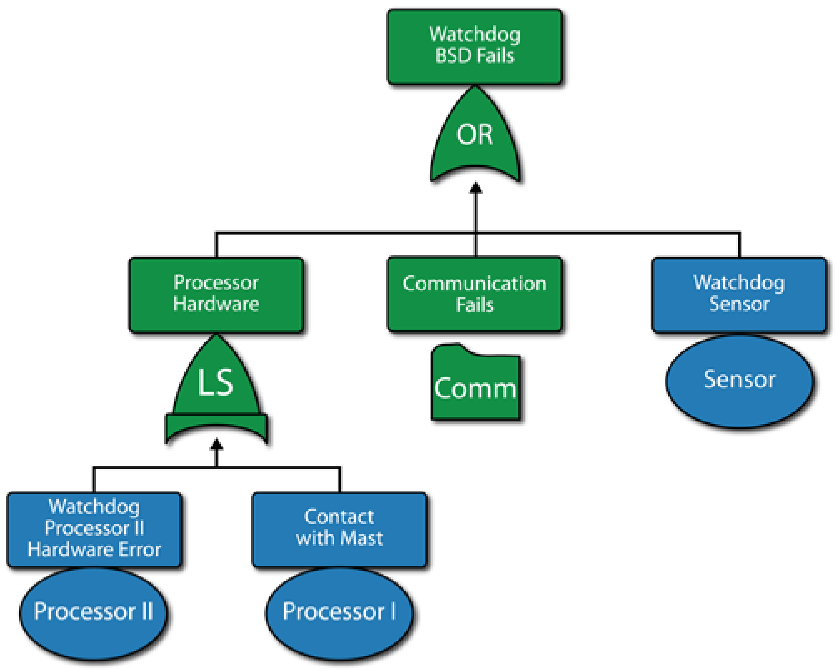
C: The fault tree of the robot arm mechanism. This subdiagram is referenced in Figure "B".



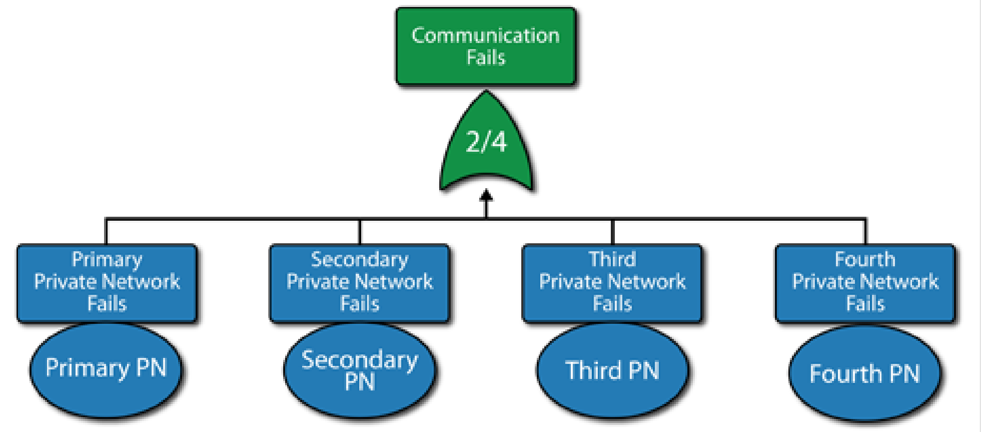
D: The fault tree for the arm jams/collides event. This subdiagram is referenced in Figure "B". It also includes a subdiagram continuation to Figure "E".



E: The brake shutdown event referenced from Figure "D". it also includes a subdiagram continuation to Figure "F".

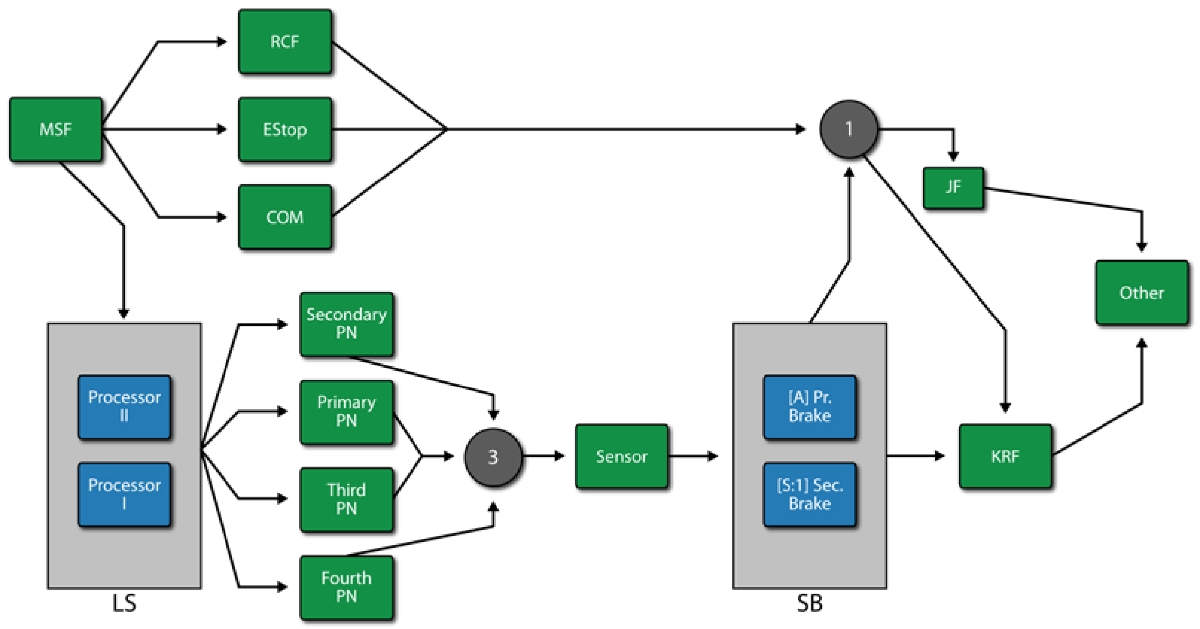


F: The watchdog ESD fails event referenced from Figure "F". It also includes a subdiagram continuation to Figure "G"

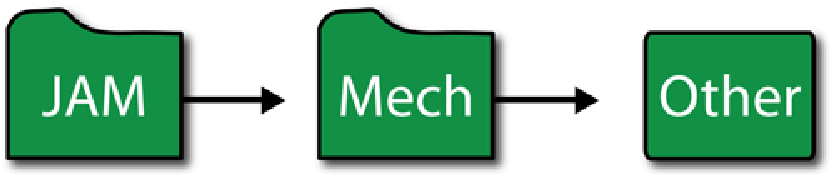

Communication Fails
2/4
Primary Private Network Fails
Secondary Private Network Fails
Third Private Network Fails
Fourth Private Network Fails
Primary PN
Secondary PN
Third PN
Fourth PN

G: The communication fails event referenced from Figure "F".

The RBD representation of the fault tree shown in the first figure is given in Figure "H". This same RBD could have been represented using subdiagrams, as shown in Figure "I". In this figure, which is the RBD representation of Figure "B", the subdiagrams in the RBD link to the fault trees of Figures "D" and "C" and their sub-trees.

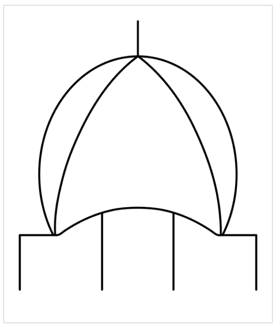


H: This is the RBD equivalent of the complete fault tree of Figure "A".



I: The RBD representation of Figure "B" with the subdiagrams in the RBD linked to the fault trees of Figures "C" and "D" and their sub-trees.

XOR Gate

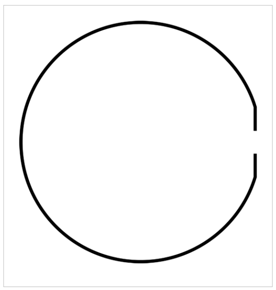


In an XOR gate, the output event occurs if exactly one input event occurs. This is similar to an OR gate with the exception that if more than one input event occurs then the output event does not occur. For example, if there are two input events then the XOR gate indicates that the output event occurs if only one of the input events occurs but not if zero or both of these events occur. From a system reliability perspective, this would imply that a two-component system would function even if both components had failed. Furthermore, when dealing with time-varying failure distributions, and if system components do not operate through failure, a failure occurrence of both components at the exact same time ($dt)$is an unreachable state; thus an OR gate would suffice. For these reasons, an RBD equivalent of an XOR gate is not presented here and BlockSim does not explicitly provide an XOR gate.

Event Classifications

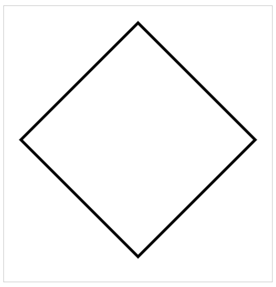
Traditional fault trees use different shapes to represent different events. Unlike gates, however, different events in a fault tree are not treated differently from an analytical perspective. Rather, the event shapes are used to convey additional information visually. BlockSim includes some of the main event symbols from classical fault tree analysis and provides utilities for changing the graphical look of a block to illustrate a different type of event. Some of these event classifications are given next. From a properties perspective, all events defined in BlockSim can have fixed probabilities, failure distributions, repair distributions, crews, spares, etc. In other words, fault tree event blocks can have all the properties that an RBD block can have. This is an enhancement and a significant expansion over traditional fault trees, which generally include just a fixed probability of occurrence and/or a constant failure rate.

Basic Event



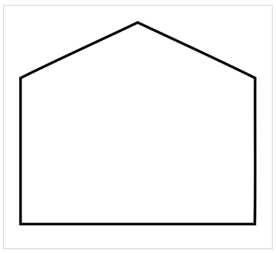
A basic event (or failure event) is identical to an RBD block and has been traditionally represented by a circle.

Undeveloped Event



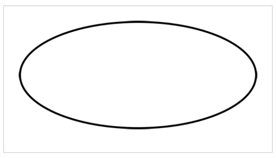
An undeveloped event has the same properties as a basic event with the exception that it is graphically rendered as a diamond. The diamond representation graphically illustrates that this event could have been expanded into a separate fault tree but was not. In other words, the analyst uses a different symbol to convey that the event could have been developed (broken down) further but he/she has chosen not to do so for the analysis.

House Event

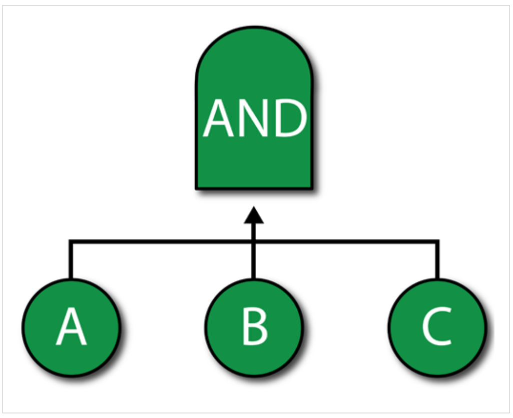


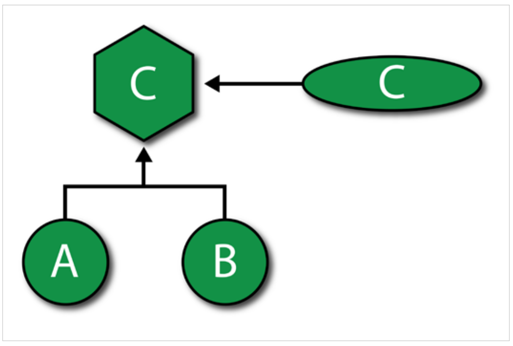
A house event is an event that can be set to occur or not occur (i.e., it usually has a fixed probability of 0 or 1). It is usually used to turn paths on or off or to make paths of a tree functional or non-functional. Furthermore, the terms failed house and working house have been used to signify probabilities of 0 and 1 respectively. In BlockSim, a house shape is available for an event and a house event has the same properties as a basic event, keeping in mind that an event can be set to Cannot Fail or Failed from the block properties window.

Conditional Event



A conditional event is represented by an ellipse and specifies a condition. Again, it has all the properties of a basic event. It can be applied to any gate. As an example, event Cin the first figure below would be the conditional event and it would be represented more applicably by an ellipse than a circle, as shown in the second figure below.





Comparing Fault Trees and RBDs

The most fundamental difference between fault tree diagrams and reliability block diagrams is that you work in the success space in an RBD while you work in the failure space in a fault tree. In other words, the RBD considers success combinations while the fault tree considers failure combinations. In addition, fault trees have traditionally been used to analyze fixed probabilities (i.e., each event that comprises the tree has a fixed probability of occurring) while RBDs may include time-varying distributions for the success (reliability equation) and other properties, such as repair/restoration distributions. In general (and with some specific exceptions), a fault tree can be easily converted to an RBD. However, it is generally more difficult to convert an RBD into a fault tree, especially if one allows for highly complex configurations.

As you can see from the discussion to this point, an RBD equivalent exists for most of the constructs that are supported by classical fault tree analysis. With these constructs, you can perform the same powerful system analysis, including simulation, regardless of how you choose to represent the system thus erasing the distinction between fault trees and reliability block diagrams.

The following example demonstrates how you can model the same analysis scenario using either RBDs or fault trees in BlockSim. The results will be the same with either approach. This discussion presents the RBD and fault tree solutions together so you can compare the methods. As an alternative, you could also review RBD Model [1] and Fault Tree Model [1], which present the steps for each modeling approach separately.

Problem Statement

Assume that a component can fail due to six independent primary failure modes: A, B, C, D, E and F. Some of these primary modes can be broken down further into the events that can cause them, or sub-modes. Furthermore, assume that once a mode occurs, the event also occurs and the mode does not go away. Specifically:

- The component fails if mode A, B or C occurs.
- If mode D, E or F occurs alone, the component does not fail; however, the component will fail if any two (or more) of these modes occur (i.e., D and E ; D and F ; E and F).
- Modes D, E and F have a constant rate of occurrence (exponential distribution) with mean times of occurrence of 200,000, 175,000 and 500,000 hours, respectively.
- The rates of occurrence for modes A, B and C depend on their sub-modes.

Do the following:

1. Determine the reliability of the component after 1 year (8,760 hours).
2. Determine the B10 life of the component.
3. Determine the mean time to failure (MTTF) of the component.
4. Rank the modes in order of importance at 1 year.
5. Recalculate results 1, 2 and 3 assuming mode B is eliminated.

To begin the analysis, modes A, B and C can be broken down further based on specific events (sub-modes), as defined next.

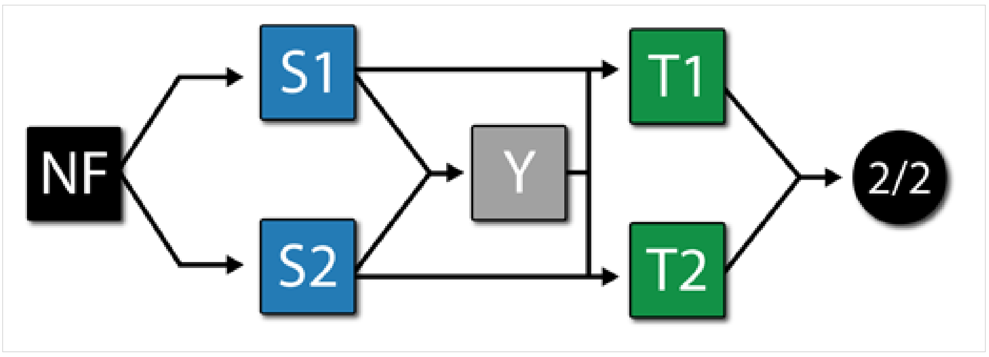
Mode A

There are five independent events (sub-modes) associated with mode A : events S1, S2, T1, T2 and Y. It is assumed that events S1 and S2 each have a constant rate of occurrence with a probability of occurrence in a single year (8,760 hours) of 1 in 10,000 and 1 in 20,000, respectively. Events T1 and T2 are more likely to occur in an older component than a newer one (i.e., they have an increasing rate of occurrence) and have a probability of occurrence of 1 in 10,000 and 1 in 20,000, respectively, in a single year and 1 in 1,000 and 1 in 3,000, respectively, after two years. Event Y also has a constant rate of occurrence with a probability of occurrence of 1 in 1,000 in a single year. There are three possible ways for mode A to manifest itself:

1. Events S1 and S2 both occur.
2. Event T1 or T2 occurs.
3. Event Y and either event S1 or event S2 occur (i.e., events Y and S1 or events Y and S2 occur).

RBD Solution for Mode A

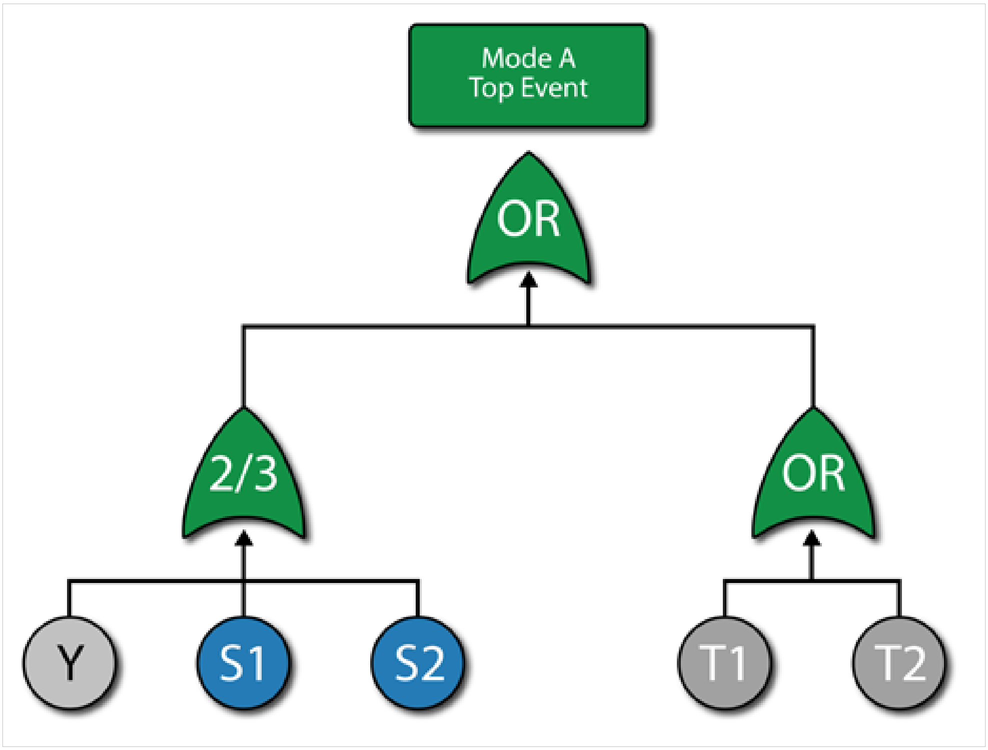
The RBD that satisfies the conditions for mode A is shown in the figure below.



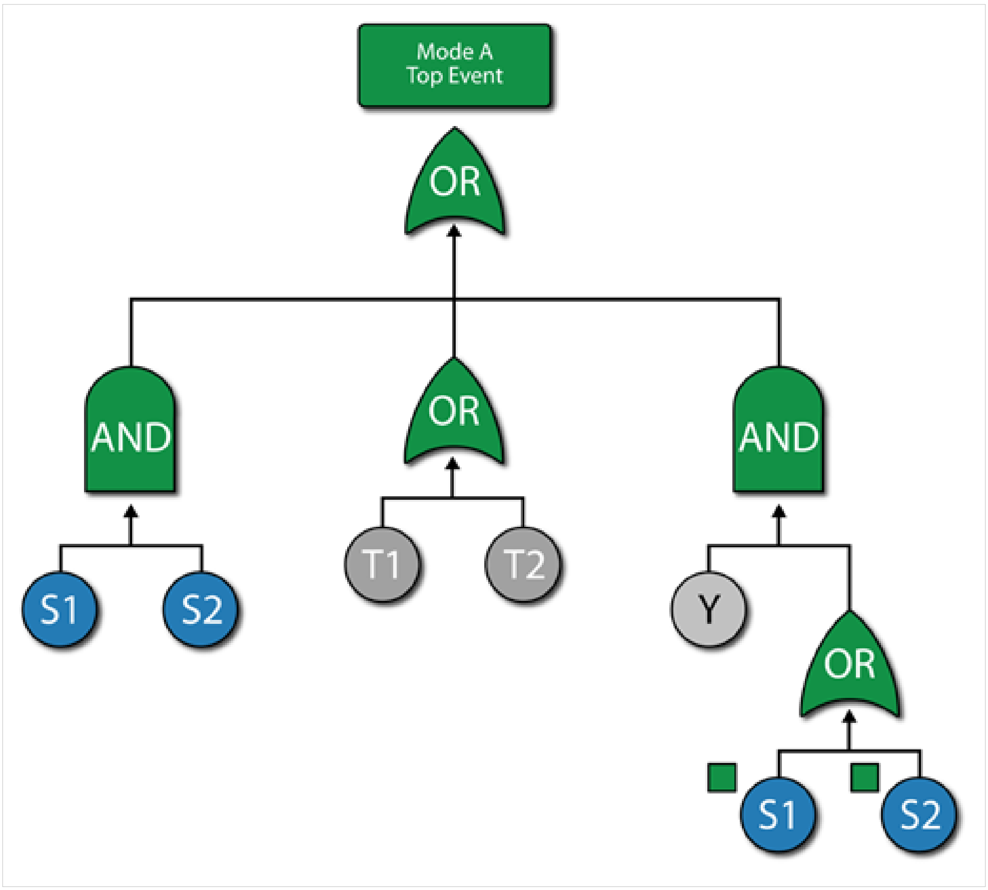
Each mode is identified in the RBD. Furthermore, two additional items are included: a starting block (NF) and an end node (2/2). The starting block and the end node are set so they cannot fail and, therefore, will not affect the results. The end node is used to define a 2-out-of-2 configuration (i.e., both paths leading into the node must work).

Fault Tree Solution for Mode A

The fault tree for mode A is shown in the figure below.



Each mode is identified as an event in the fault tree. The following figure shows an alternative representation for mode A using mirrored events for S1 and S2.

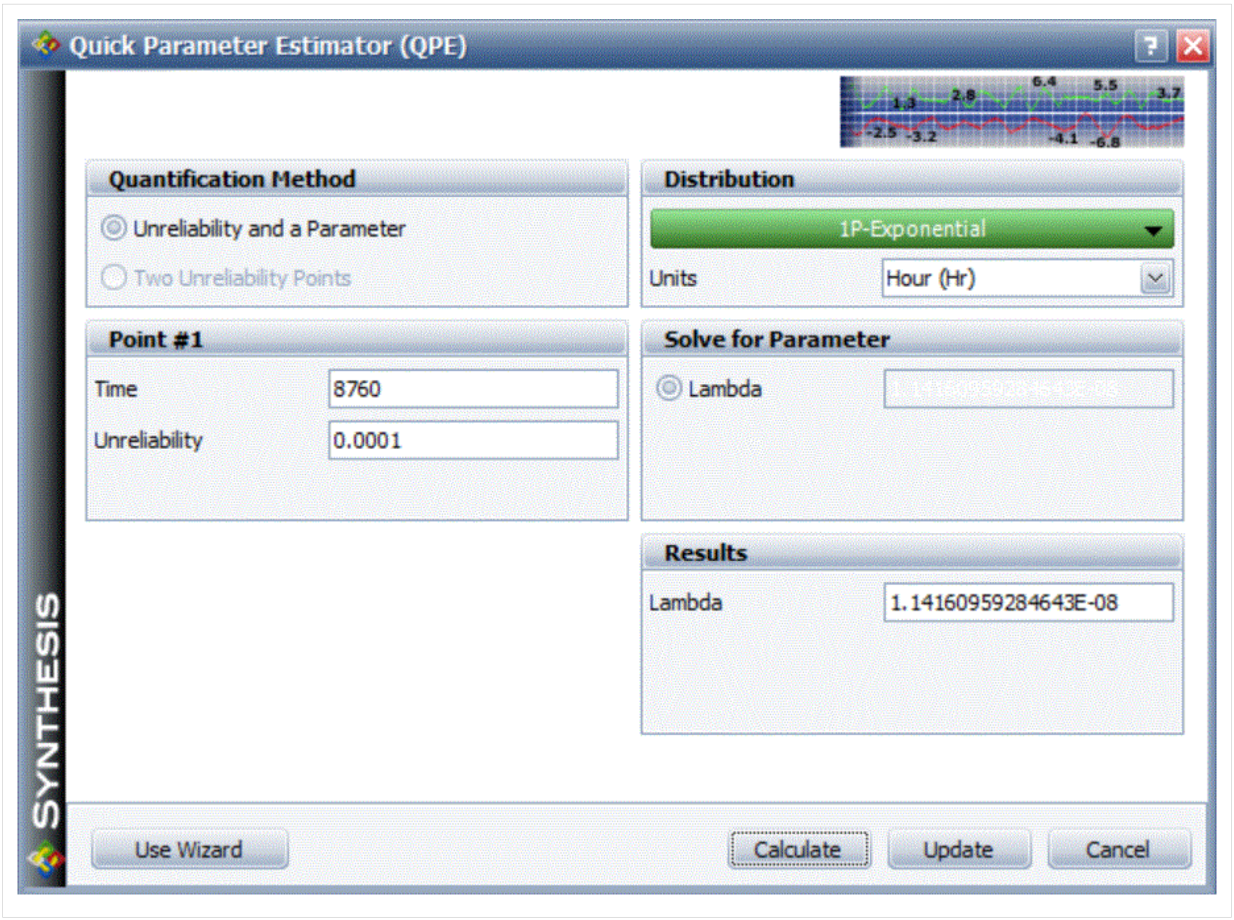


Mode A Discussion

The system reliability equation for this configuration (regardless of how it is drawn) is:

$$\begin{aligned} R(t) = & -2R_{T2}\cdot R_{S1}\cdot R_{S2}\cdot R_{T1}\cdot R_{Y} \\ & +R_{T2}\cdot R_{S1}\cdot R_{S2}\cdot R_{T1} \\ & +R_{T2}\cdot R_{S1}\cdot R_{T1}\cdot R_{Y} \\ & +R_{T2}\cdot R_{S2}\cdot R_{T1}\cdot R_{Y} \end{aligned}$$

Based on the given probabilities, distribution parameters are computed for each block (either RBD block or the fault tree event block). One way is to compute them using the Parameter Experimenter, as shown in the figure below. In this figure and for S1, the probability is 1 in 10,000 in one year (8,760 hours), thus the exponential failure rate is 1.1416e-8. This can be repeated for S2 and Y.


Quick Parameter Estimator (QPE)

Quantification Method: [✓ Unreliability and a Parameter] Two Unreliability Points

Distribution: 1P-Exponential

Units: Hour (Hr)

Point #1

Time: 8760

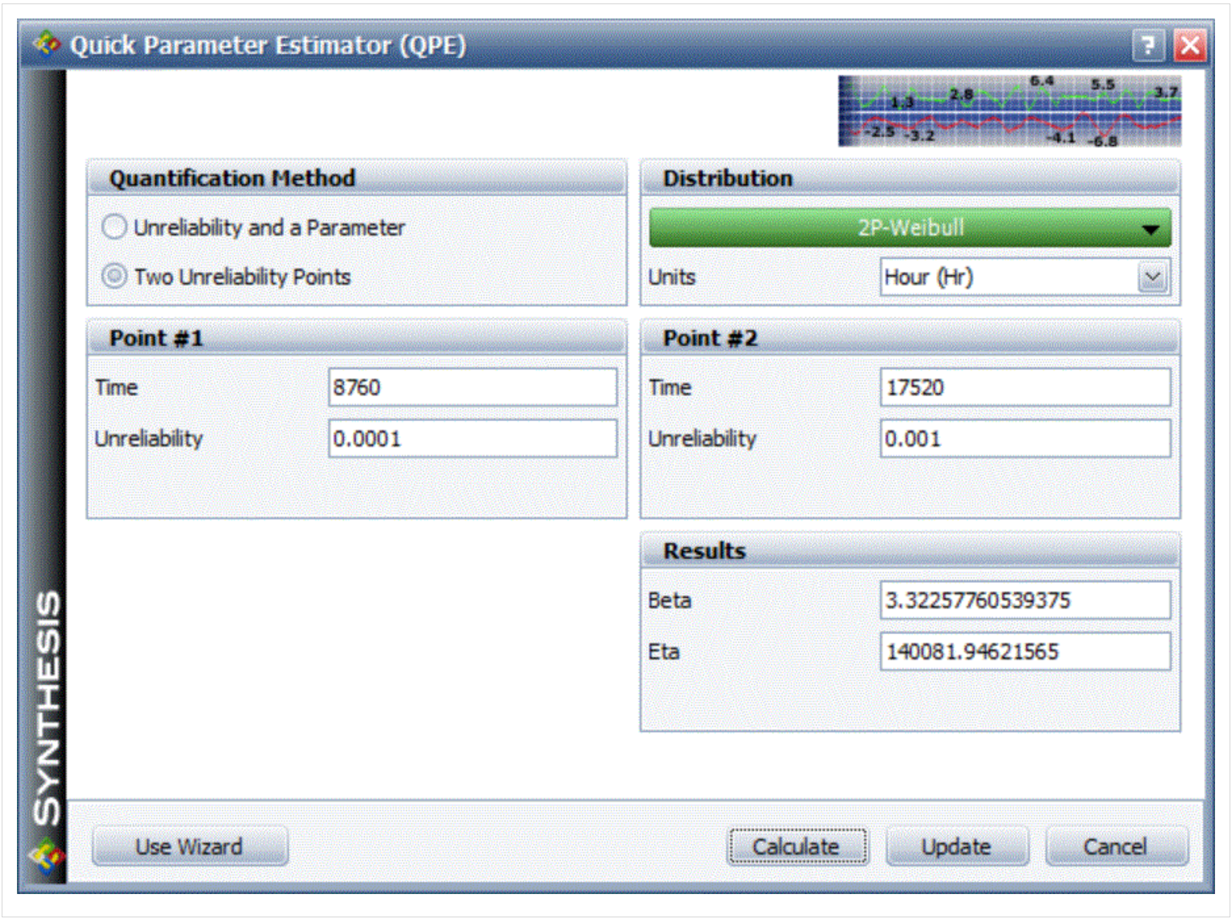
Unreliability: 0.0001

Solve for Parameter: [✓ Lambda]

Results

Lambda: 1.14160959284643E-08

Events T1 and T2 need to be modeled using a life distribution that does not have a constant failure rate. Using BlockSim's Parameter Experimenter and selecting the Weibull distribution, the parameter values for events T1 and T2 are shown in the figures below.



Quantification Method

Unreliability and a Parameter

[✓ Two Unreliability Points]

Distribution: 2P-Weibull

Units	Hour (Hr)

Point #1

Time	8760
Unreliability	0.0001

Point #2

Time	17520
Unreliability	0.001

Results

Beta	3.32257760539375
Eta	140081.94621565

Mode B

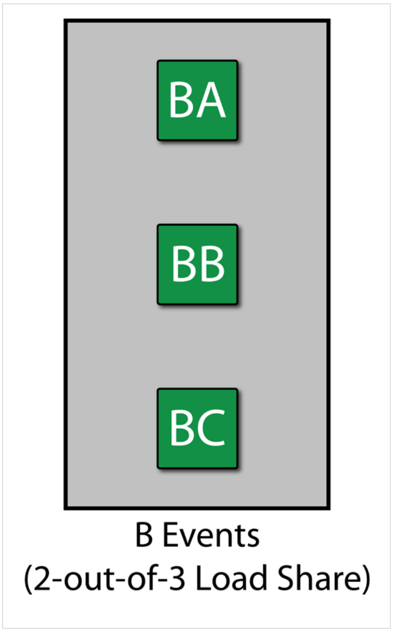
There are three dependent events associated with mode B : events BA, BB and BC.

- Two out of the three events must occur for mode B to occur.
    - o Events BA, BB and BC all have an exponential distribution with a mean of 50,000 hours.
    - o The events are dependent (i.e., if BA, BB or BC occurs, then the remaining events are more likely to occur). Specifically, when one event occurs, the MTTF of the remaining events is halved.

This is basically a load sharing configuration. The reliability function for each block will change depending on the other events. Therefore, the reliability of each block is not only dependent on time, but also on the stress (load) that the block experiences.

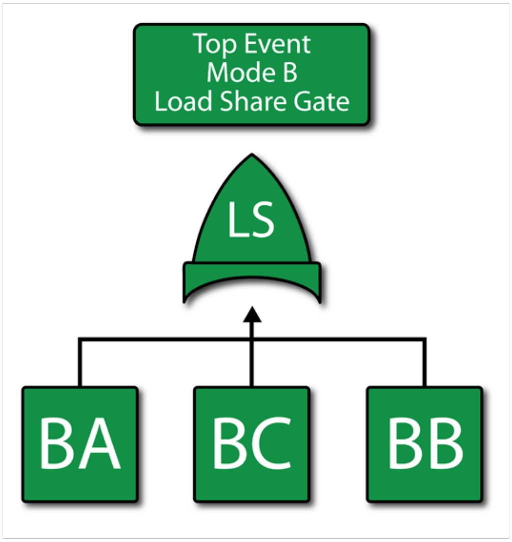
RBD Solution for Mode B

The reliability block diagram for mode B is shown in the figure below.



Fault Tree Solution for Mode B

The fault tree for mode B is shown in the figure below. A Load Sharing gate is used.



Mode B Discussion

To describe the dependency, a Load Sharing gate and dependent event blocks are used. Since the failure rate is assumed to be constant, an exponential distribution is used. Furthermore, for simplicity, an Arrhenius life-stress relationship is used with the parameters B=2.0794 and C=6250.

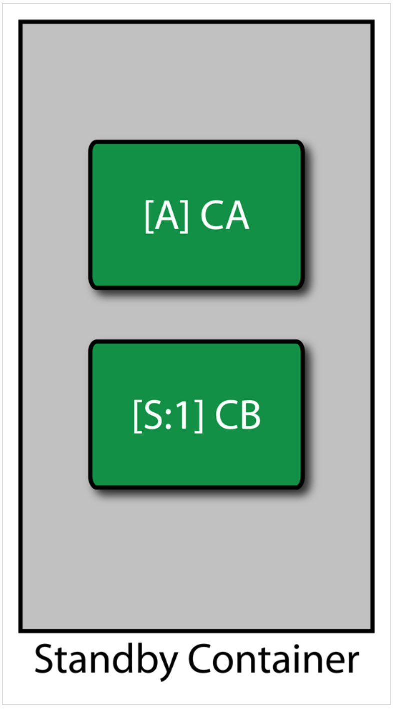
Mode C

There are two sequential events associated with mode C : CA and CB.

- Both events must occur for mode C to occur.
- Event CB will only occur if event CA has occurred.
- If event CA has not occurred, then event CB will not occur.
- Events CA and CB both occur based on a Weibull distribution.
- For event CA, β= 2 and η= 30,000 hours.
- For event CB, β= 2 and η= 10,000 hours.

RBD Solution for Mode C

To model this, you can think of a scenario similar to standby redundancy. Basically, if CA occurs then CB gets initiated. A Standby container can be used to model this, as shown in the figure below.



In this case, event CA is set as the active component and CB as the standby. If event CA occurs, CB will be initiated. For this analysis, a perfect switch is assumed. The properties are set in BlockSim as follows:

Contained Items

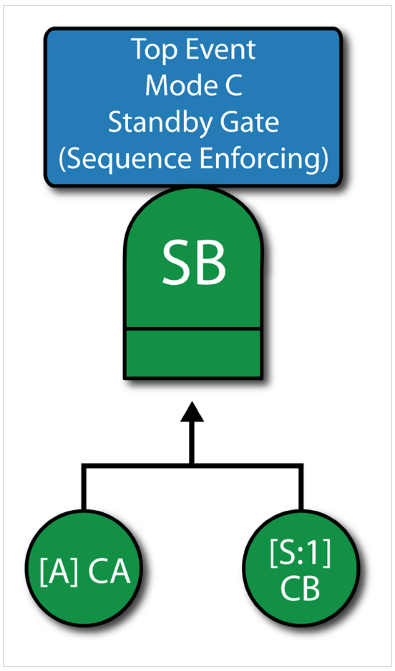
- CA : Active failure distribution, Weibull distribution (β= 2, η= 30,000).
- CA : Quiescent failure distribution: None, cannot fail or age in this mode.
- CB : Active failure distribution, Weibull distribution (β= 2, η= 10,000).
- CB : Quiescent failure distribution: None, cannot fail or age in this mode.

Switch

- Active Switching: Always works (100% reliability) and instant switch (no delays).
- Quiescent Switch failure distribution: None, cannot fail or age in this mode.

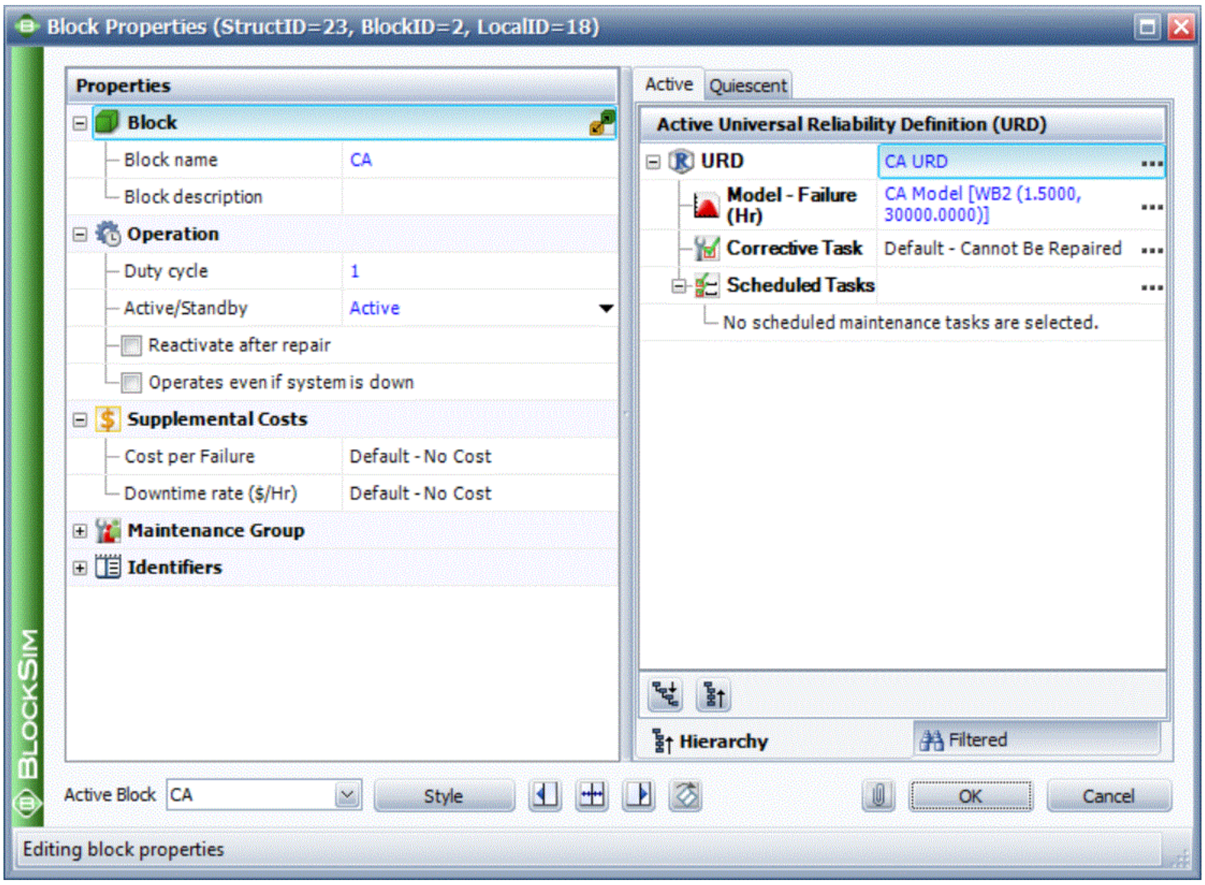
Fault Tree Solution for Mode C

The fault tree for mode C is shown in the figure below. Note that the sequence is enforced by the Standby gate (used as a Sequence Enforcing gate).



Mode C Discussion

The failure distribution settings for event CA are shown in the figure below.


Block Properties (StructID=23, BlockID=2, LocalID=18)

Properties

Block
- Block name: CA
- Block description:

Operation
- Duty cycle: 1
- Active/Standby: Active
- Reactivate after repair: ☐
- Operates even if system is down: ☐

Supplemental Costs
- Cost per Failure: Default - No Cost
- Downtime rate ($/Hr): Default - No Cost

Maintenance Group

Identifiers

Active | Quiescent

Active Universal Reliability Definition (URD)
- URD: CA URD
- Model - Failure (Hr): CA Model [WB2 (1.5000, 30000.0000)]
- Corrective Task: Default - Cannot Be Repaired
- Scheduled Tasks:
  - No scheduled maintenance tasks are selected.

Hierarchy | Filtered

Active Block: CA

Editing block properties

The failure distribution properties for event CB are set in the same manner.

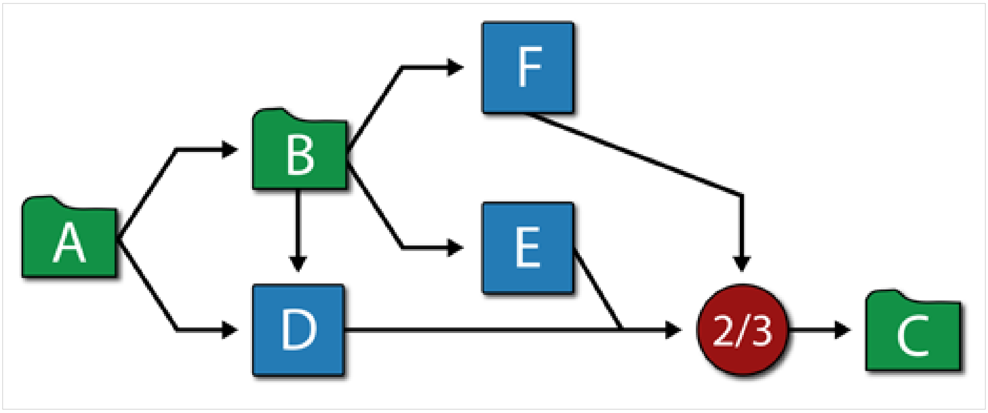
Modes D, E and F

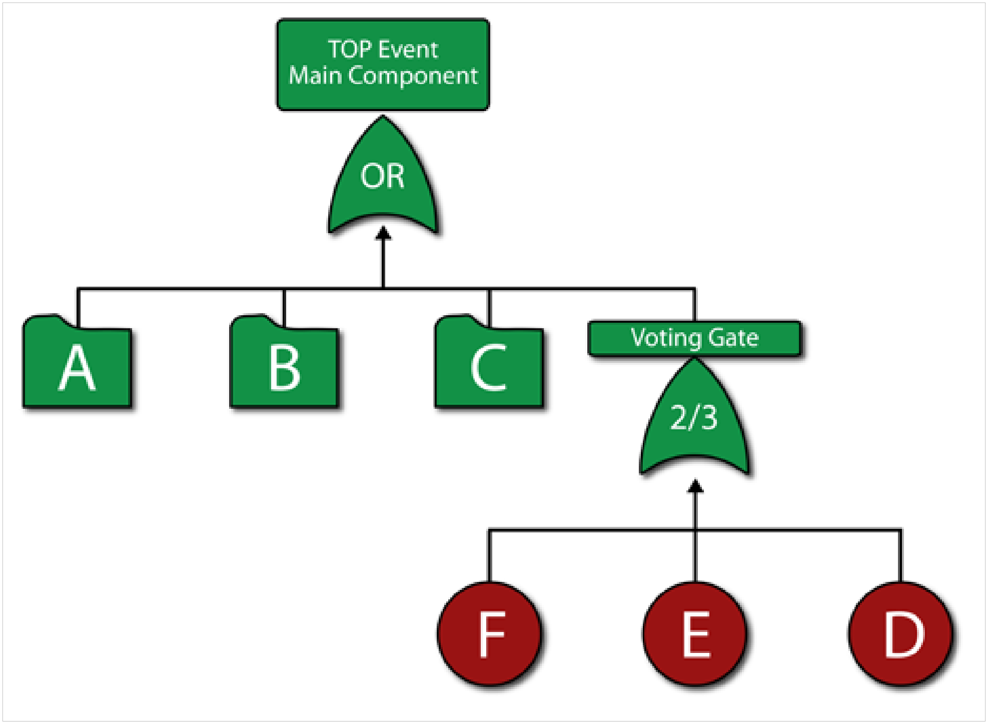
Modes D, E and F can all be represented using the exponential distribution. The failure distribution properties for modes D, E and F are:

- D : MTTF = 200,000 hours.
- E : MTTF = 175,000 hours.
- F : MTTF = 500,000 hours.

The Entire Component

The last step is to set up the model for the component based on the primary modes (A, B, C, D, E and F). Modes A, B and C can each be represented by single blocks that encapsulate the subdiagrams already created. The RBD in the first figure below represents the primary failure modes for the component while the fault tree in second figure below illustrates the same. The node represented by 2/3 in the RBD indicates a 2-out-of-3 configuration. The Voting OR gate in the fault tree accomplishes the same. Subdiagrams are used in both configurations for the sub-modes.





Once the diagrams have been created, the reliability equation for the system can be obtained, as follows:

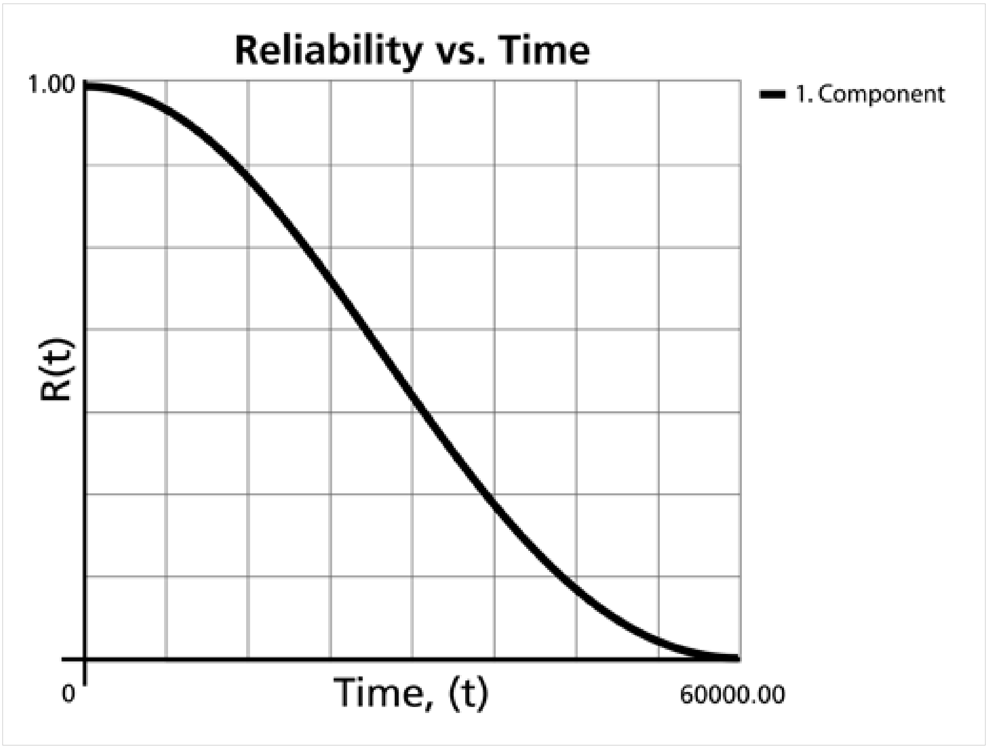
$$\begin{aligned}R(t)_{System} =& R(t)_A \cdot R(t)_B \cdot R(t)_F \cdot R(t)_D \cdot R(t)_C \\ &+ R(t)_A \cdot R(t)_B \cdot R(t)_F \cdot R(t)_C \cdot R(t)_E \\ &+ R(t)_A \cdot R(t)_B \cdot R(t)_D \cdot R(t)_C \cdot R(t)_E \\ &- 2(R(t)_A \cdot R(t)_B \cdot R(t)_F \cdot R(t)_D \cdot R(t)_C \cdot R(t)_E)\end{aligned}$$

where $R(t)_A$, $R(t)_B$and $R(t)_C$are the reliability equations corresponding to the sub-modes.

Analysis and Discussion

The questions posed earlier can be answered using BlockSim. Regardless of the approach used (i.e., RBD or FTA), the answers are the same.

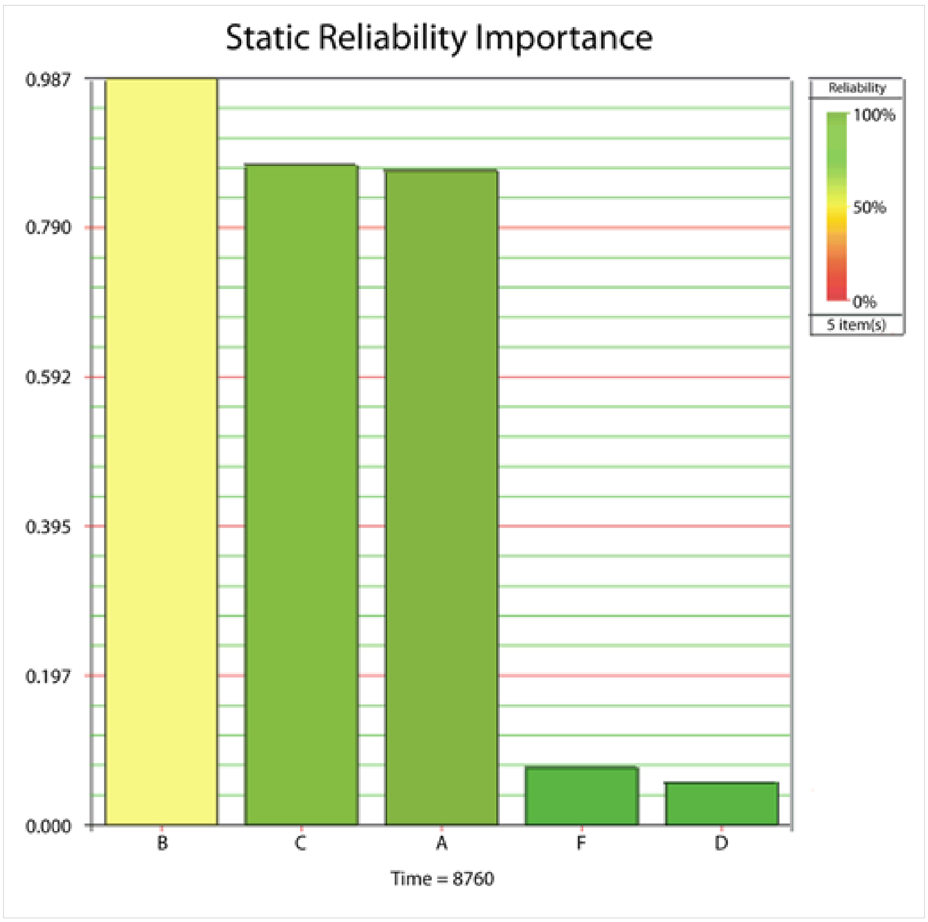
1. The reliability of the component at 1 year (8,760 hours) can be calculated using the Analytical Quick Calculation Pad (QCP) or by viewing the reliability vs. time plot, as displayed in the following figure. $R(t = 8760) = 86.4975\%$.



2. Using the Analytical QCP, the B10 life of the component is estimated to be 7,373.94 hours.

3. Using the Analytical QCP, the mean life of the component is estimated to be 21,659.68 hours.

4. The ranking of the modes after 1 year can be shown via the Static Reliability Importance plot, as shown in the figure below.



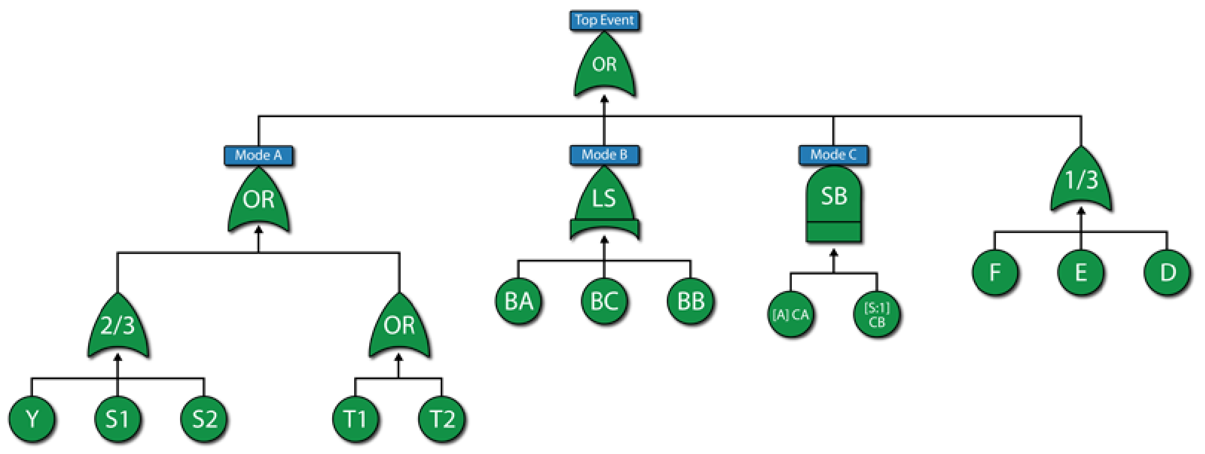
5. Re-computing the results for 1, 2 and 3 assuming mode B is removed:

   a) R(t=8760) =98.72%.

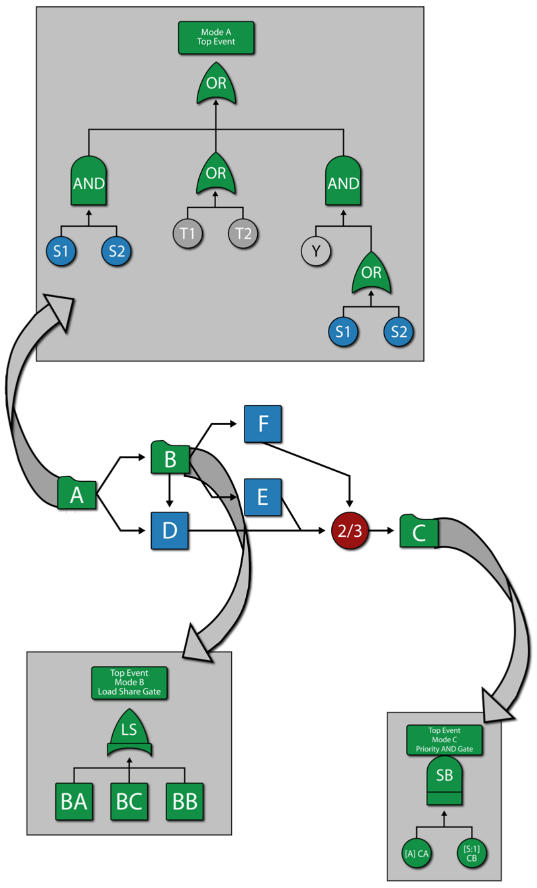
   b) B10 = 16,928.38 hours.

   c) MTTF = 34,552.89 hours.

There are multiple options for modeling systems with fault trees and RBDs in BlockSim. The first figure below shows the complete fault tree for the component without using subdiagrams (Transfer gates) while the second figure below illustrates a hybrid analysis utilizing an RBD for the component and fault trees as the subdiagrams. The results are the same regardless of the option chosen.



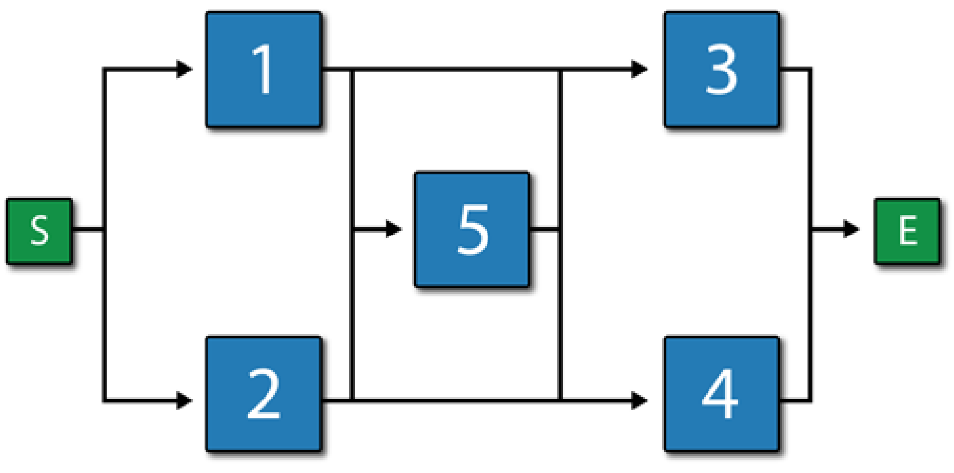
Fault tree for the component without using subdiagrams (Transfer gates)



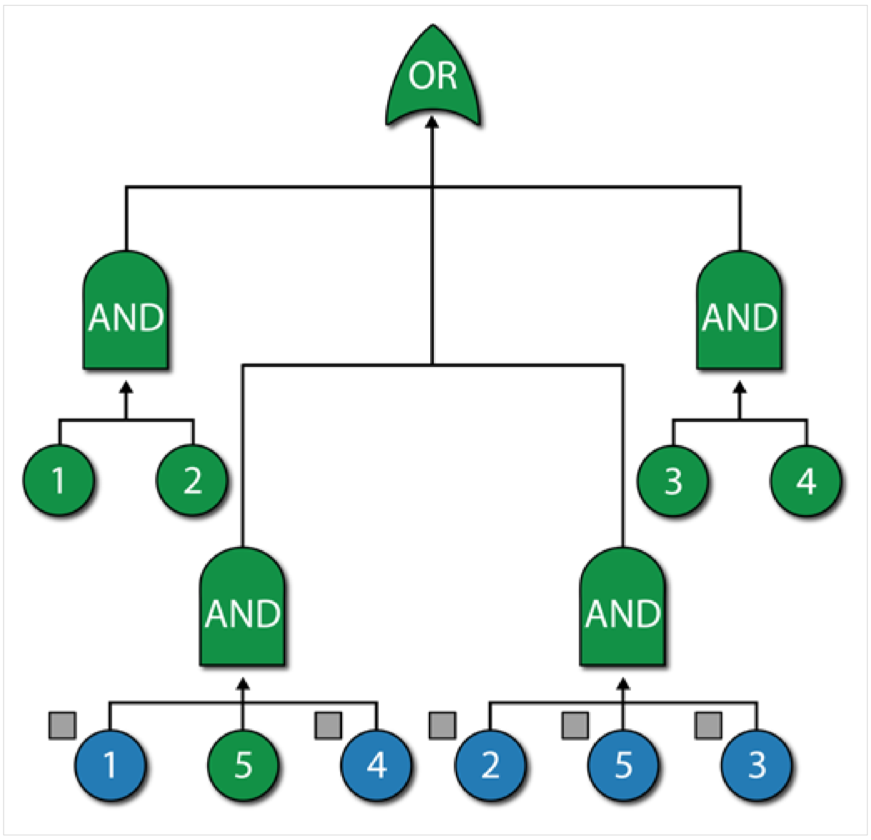
A hybrid solution using an RBD for the component and fault trees as subdiagrams.

Using Mirrored Blocks to Represent Complex RBDs as FTDs

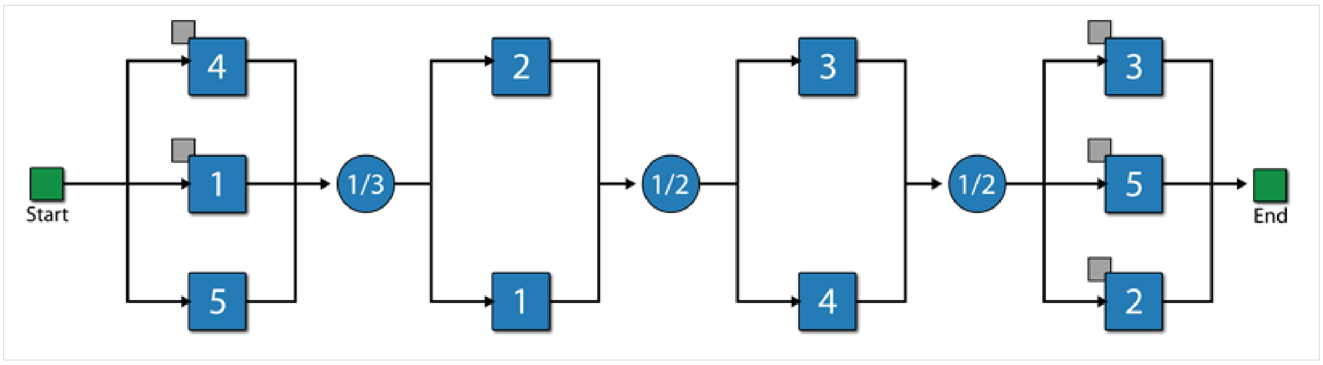
A fault tree cannot normally represent a complex RBD. As an example, consider the RBD shown in the figure below.



A fault tree representation for this RBD is:



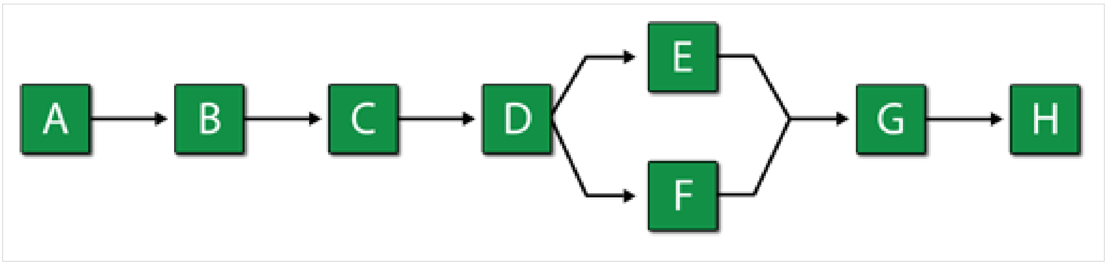
Note that the same event is used more than once in the fault tree diagram. To correctly analyze this, the duplicate events need to be set up as "mirrored" events to the parent event. In other words, the same event is represented in two locations in the fault tree diagram. It should be pointed out that the RBD in the following figure is also equivalent to the RBD shown earlier and the fault tree of the figure shown above.

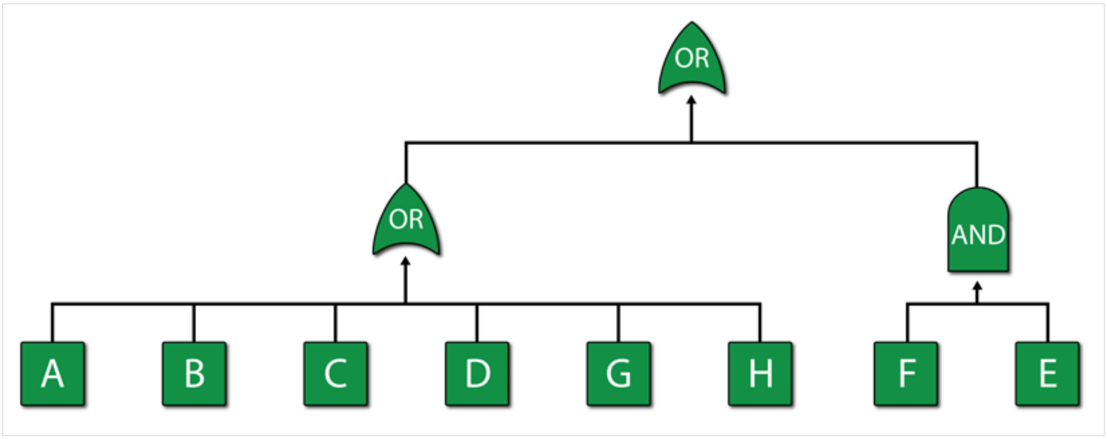


Fault Trees and Simulation

The slightly modified constructs in BlockSim erase the distinction between RBDs and fault trees. Given this, any analysis that is possible in a BlockSim RBD (including throughput analysis) is also available when using fault trees.

As an example, consider the RBD shown in the first figure below and its equivalent fault tree representation, as shown in the second figure.





Furthermore, assume the following basic failure and repair properties for each block and event:

- Block *A*:
  - o Failure Distribution: Weibull; $\beta = 1/5$; $\eta = 1,000$.
  - o Corrective Distribution: Weibull; $\beta = 1.5$; $\eta = 100$.
- Block *B*:
  - o Failure Distribution: Exponential; $\mu = 10,000$.
  - o Corrective Distribution: Weibull; $\beta = 1.5$; $\eta = 20$.
- Block *C*:
  - o Failure Distribution: Normal; $\mu = 1,000$; $\sigma = 200$.
  - o Corrective Distribution: Normal; $\mu = 6$; $\sigma = 2$.
- Block *D*:
  - o Failure Distribution: Weibull; $\beta = 1.5$; $\eta = 10,000$.
  - o Corrective Distribution: Exponential; $\mu = 10$.
- Block *E*:
  - o Failure Distribution: Weibull; $\beta = 3$; $\eta = 1,000$.
  - o Corrective Distribution: Weibull; $\beta = 1.5$; $\eta = 20$.
- Block *F*:
  - o Failure Distribution: Weibull; $\beta = 1.5$; $\eta = 5,000$.

o Corrective Distribution: Weibull; $\beta = 1.5$; $\eta = 100$.

- Block *G*:

  o Failure Distribution: Exponential; $\mu = 100,000$.

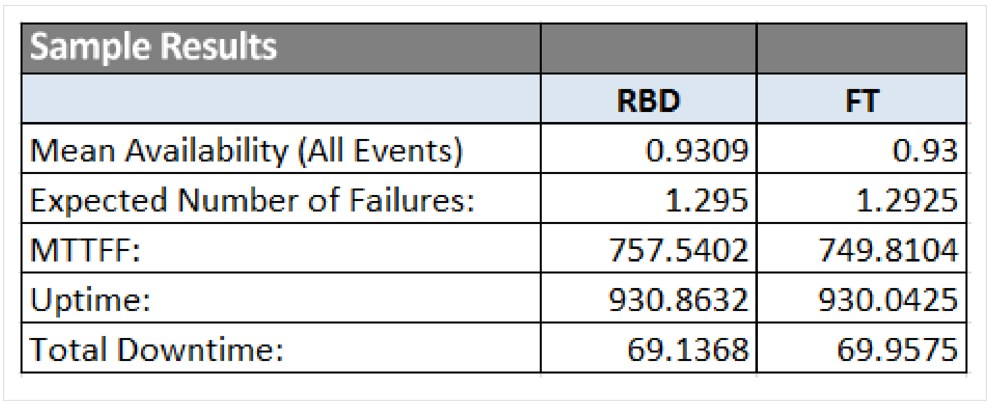
  o Corrective Distribution: Weibull; $\beta = 1.5$; $\eta = 10$.

- Block *H*:

  o Failure Distribution: Normal; $\mu = 5,000$; $\sigma = 50$.

  o Corrective Distribution: Normal; $\mu = 10$; $\sigma = 2$.

A sample table of simulation results is given next for up to $t = 1,000$, using $2,000$simulations for each diagram and an identical seed.



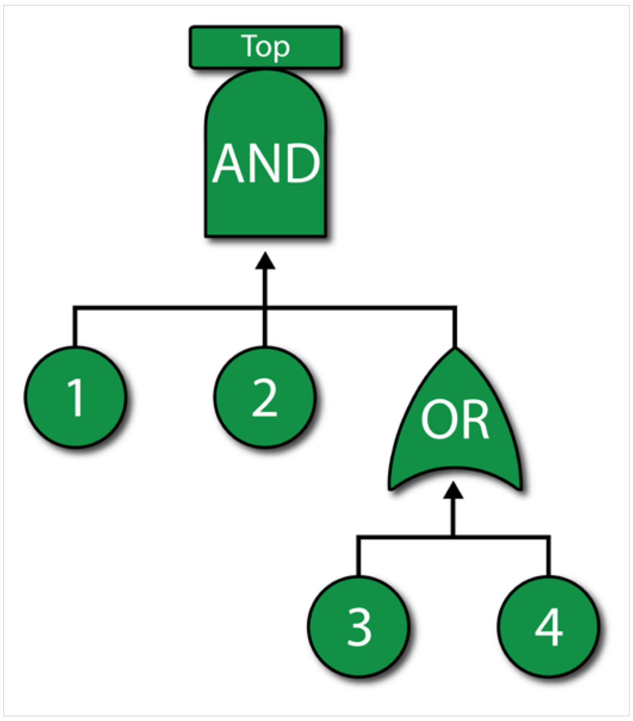
Sample Results		
	RBD	FT
Mean Availability (All Events)	0.9309	0.93
Expected Number of Failures:	1.295	1.2925
MTTFF:	757.5402	749.8104
Uptime:	930.8632	930.0425
Total Downtime:	69.1368	69.9575

As expected, the results are equivalent (within an expected difference due to simulation) regardless of the diagram type used. It should be pointed out that even though the same seed was used by both diagrams, the results are not always expected to be identical because the order in which the blocks are read from a fault tree diagram during the simulation may differ from the order in which they are read in the RBD; thus using a different random number stream for each block (e.g., block *G* in the RBD may receive a different sequence of random numbers than event block *G* in the FT).

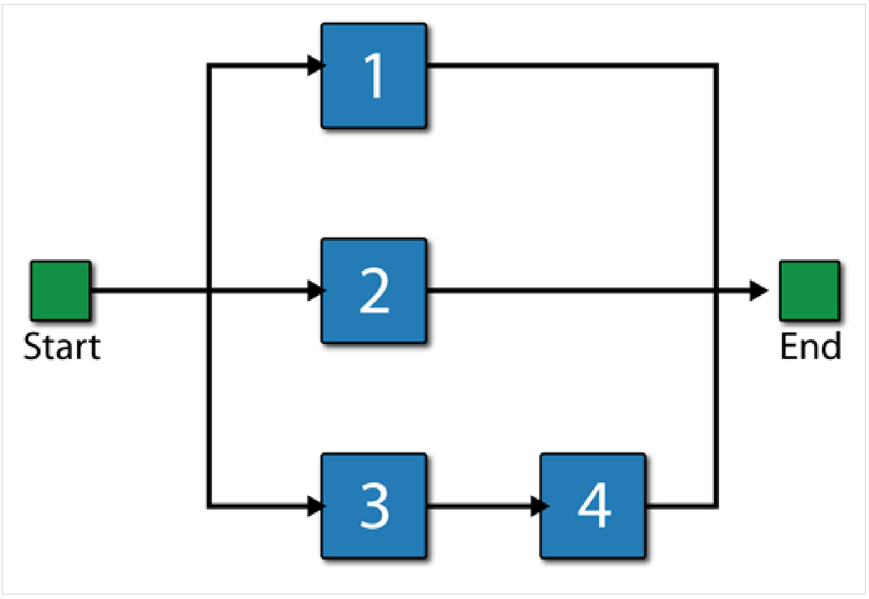
Additional Fault Tree Topics

Minimal Cut Sets

Traditional solution of fault trees involves the determination of so-called *minimal cut sets*. Minimal cut sets are all the unique combinations of component failures that can cause system failure. Specifically, a cut set is said to be a minimal cut set if, when any basic event is removed from the set, the remaining events collectively are no longer a cut set, as discussed in Kececioglu [10]. As an example, consider the fault tree shown in the figure below. The system will fail if {1, 2, 3 and 4 fail} or {1, 2 and 3 fail} or {1, 2 and 4 fail}.



All of these are cut sets. However, the one including all components is not a minimal cut set because, if 3 and 4 are removed, the remaining events are also a cut set. Therefore, the minimal cut sets for this configuration are {1, 2 , 3} or {1, 2, 4}. This may be more evident by examining the RBD equivalent of the figure above, as shown in the figure below.



BlockSim does not use the cut sets methodology when analyzing fault trees. However, interested users can obtain these cut sets for both fault trees and block diagrams with the command available in the Analysis Ribbon.

References

[1] http://www.reliasoft.com/BlockSim/examples/rc6/index.htm

Chapter 10

Reliability Phase Diagrams (RPDs)

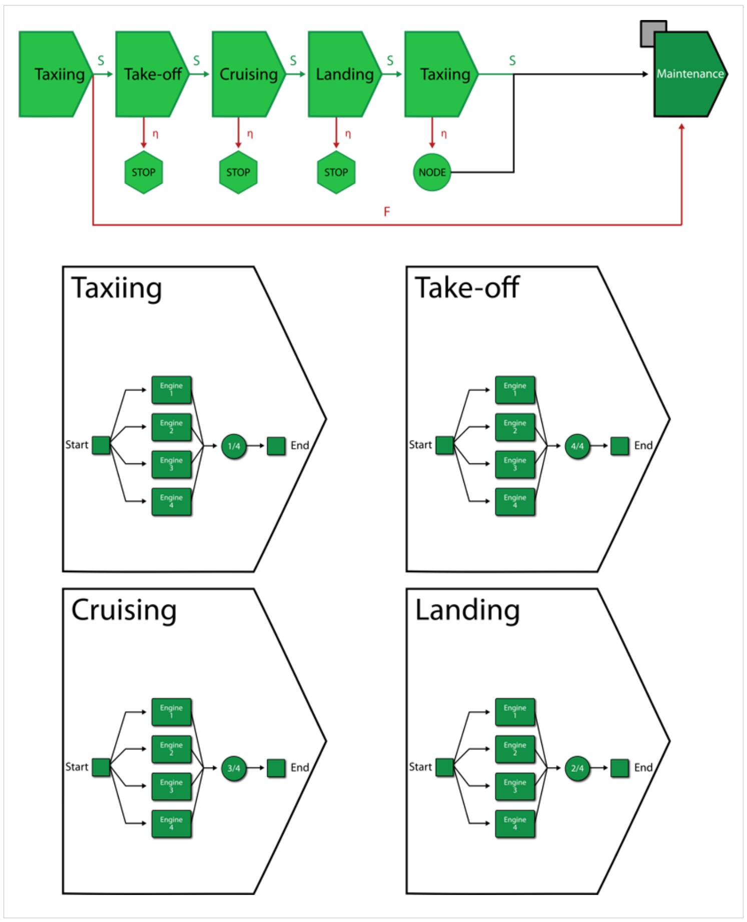
The term *phase diagram* is used in many disciplines with different meanings. In physical chemistry, mineralogy and materials science, a phase diagram is a type of graph used to show the equilibrium conditions among the thermodynamically-distinct phases. In mathematics and physics, a phase diagram is used as a synonym for a phase space. In reliability engineering, we introduce the term phase diagram, or more specifically Reliability Phase Diagram or RPD, as an extension of the reliability block diagram (RBD) approach to graphically describe the sequence of different operational and/or maintenance phases experienced by a system. Whereas a reliability block diagram (RBD) is used to analyze the reliability of a system with a fixed configuration, a phase diagram can be used to represent/analyze a system whose reliability configuration and/or other properties change over time. In other words, during a mission the system may undergo changes in its reliability configuration (RBD), available resources or the failure, maintenance and/or throughput properties of its individual components. Examples of this include:

1. Systems whose components exhibit different failure distributions depending on changes in the stress on the system.
2. Systems or processes requiring different equipment to function over a cycle, such as start-up, normal production, shut-down, scheduled maintenance, etc.
3. Systems whose RBD configuration changes at different times, such as the RBD of the engine configuration on a four-engine aircraft during taxi, take-off, cruising, and landing.
4. Systems with different types of machinery operating during day and night shifts and with different amounts of throughput during each shift.

To analyze such systems, each stage during the mission can be represented by a phase whose properties are inherited from an RBD corresponding to that phase's reliability configuration, along with any associated resources of the system during that time. A phase diagram is then a series of such phases drawn (connected) in a sequence signifying a chronological order.

To better illustrate this, consider the four-engine aircraft mentioned previously. Assume that when a critical failure (system failure) occurs during taxiing, the airplane does not take off and is sent for maintenance instead. However, when a critical failure occurs during take-off, cruising, landing, the system is assumed to be lost. Furthermore, assume that the taxi phase requires only one engine, the take-off phase requires all four engines, the cruising phase requires any three of the four engines and the landing phase requires any two of the four engines. To model this, each one of these cases would require a different k-out-of-n redundancy on the engines and thus a different RBD. Creating an RBD for each phase is trivial. However, what you need is a way to transition from one RBD to the next, in a specified sequence, while maintaining all the past history of each component during the transition.

In other words, a new engine would transition to the take-off phase with an age equal to the time it was used during taxi, or an engine that failed while in flight would remain failed in the next phase (i.e., landing). To model this, a block for each phase would be used in the phase diagram, and each phase block would be linked to the appropriate RBD. This is illustrated in the figure below.



In this figure, the taxiing, take-off, cruising and landing blocks represent the operational phases and the final block is a maintenance phase. Each of the operational phases in this diagram has two paths leading from it: a success path and a failure path. This graphically illustrates the consequences in each case. For instance, if the first taxiing phase is successful, the airplane will proceed to the take-off phase; if it is unsuccessful, the airplane will be sent for maintenance. The failure paths for the take-off, cruising and landing phases point to stop blocks, which indicate that the simulation of the mission ends. For the final taxiing phase, both the success and failure paths lead to the maintenance path; the node block allows you to model this type of shared outcome.

The execution of a phase diagram from its first phase to its last phase is referred to as one cycle. For example, for the airplane mentioned above, a single mission from the initial taxiing through the post-flight maintenance is one cycle. If the simulation end time exceeds the total duration of one cycle of a phase diagram, the simulation continues and the phase diagram is executed multiple times until the simulation end time is reached. Execution of a phase diagram multiple times during a simulation is referred to as cycling. During cycling, the age of components accumulated in the last phase of the previous cycle is carried over to the first phase of the next cycle. The principle of cumulative damage is used to transfer the age across phases for each component (block). In summary, cycling is used to model the continuous operation of a system involving repetition of the same phases in the same sequence (e.g., an airplane making multiple flights).

The sections that follow describe the types of phase blocks and other blocks that can be used in a phase diagram, as well as how those blocks can be connected. A more detailed example is then presented, followed by a discussion of the rules and assumptions that apply in phase diagram simulation. Finally, throughput in phase diagrams is discussed.

Phase Blocks

Operational Phases

In RPDs, two types of phases are used: operational phases and maintenance phases. An operational phase is used to represent any stage of the system's mission that is not exclusively dedicated to the execution of maintenance tasks. Operational phases are always defined by (linked to) an RBD. Each operational phase has a fixed, predefined time duration.

Operational Phase Properties

Diagram The diagram property is used to associate an RBD with a phase. You can select and associate any existing simulation RBD with a phase. Note that common components across different RBDs are identified by name. In other words, a component with the exact same name in two RBDs is assumed to be the same component working in two different phases.

Phase Duration The duration of an operational phase is fixed and needs to be specified. However, this duration may be affected by the choice of path you choose followed by this phase. If a failure has not occurred by the end of the specified phase duration, the simulation will proceed along the success path leading from the phase block. If a failure occurs, the simulation will proceed along the failure path leading from the phase block.

Phase Duty Cycle This property allows you to specify a common duty cycle value for the entire RBD that the phase represents, thereby modeling situations where the actual usage of the RBD during system operation is not identical to the usage for which you have data (either from testing or from the field). This can include situations where the item:

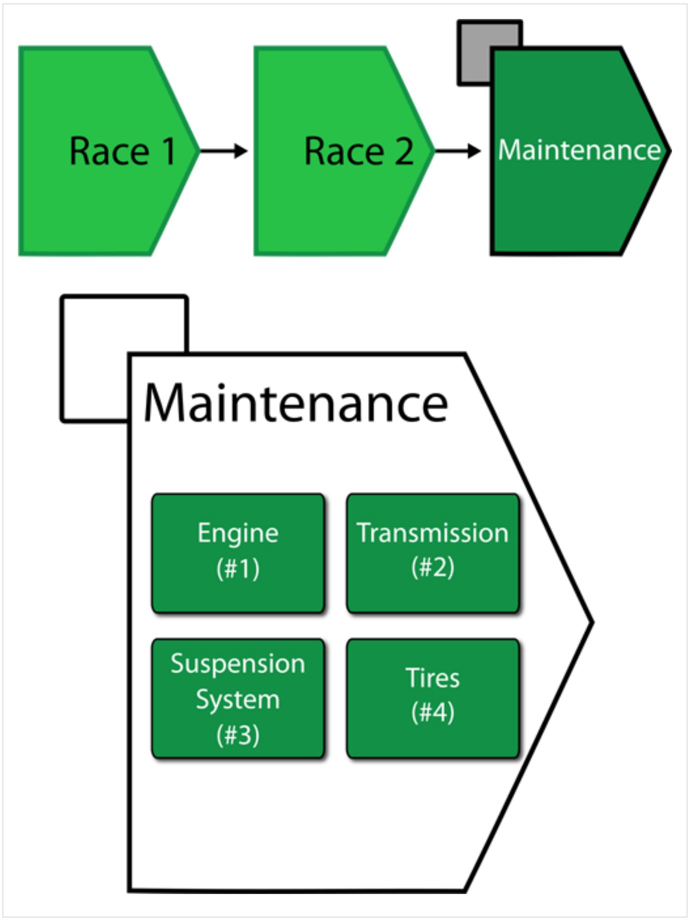
- Does not operate continuously (e.g., a DVD drive that was tested in continuous operation, but in actual use within a computer accumulates only 18 minutes of usage for every hour the computer operates).
- Is subjected to loads that are greater than or less than the rated loads (e.g., a motor that is rated to operate at 1,000 rpm but is being used at 800 rpm).
- Is affected by changes in environmental stress (e.g., a laptop computer that is typically used indoors at room temperature, but is being used outdoors in tropical conditions).

In these cases, continuous operation at the rated load is considered to be a duty cycle of 1. Any other level of usage is expressed as a percentage of the rated load value or operating time. For example, consider the DVD drive mentioned above; its duty cycle value would be 18 min / 60 min = 0.3. A duty cycle value higher than 1 indicates a load in excess of the rated value.

If a duty cycle is specified for the phase and there are also duty cycles specified for blocks within the RBD, their effects are compounded. For instance, consider the aircraft example given earlier. During the take-off phase, the subsystems experience 1.5 times the normal stress, so you would use a phase duty cycle value of 1.5. We also know that the landing gear is not used continuously during take-off. Assume that the landing gear is actually in use only 30% of the time during take-off. Each landing gear block in the RBD, then, would have a duty cycle value of 0.3. For each block, the effects of the phase duty cycle and the block duty cycle are compounded, yielding an effective duty cycle value of 1.5 x 0.3 = 0.45.

Maintenance Phases

A maintenance phase represents the portion of a system's mission time where the system is down and maintenance actions are performed on some or all of its components. For representation ease a maintenance phase is defined by (linked to) a maintenance template. This template can be thought of as a list, or a collection, of the specific components (blocks) that are designated to undergo inspection, repair or replacement actions during the maintenance phase, along with their maintenance priority order. In other words, if blocks A, B and C are to undergo maintenance during a specific phase, they are placed in a maintenance template in a priority sequence. Depending on the resources available, the actions are prioritized as resources permit. That is, if three repair crews were available along with three spare parts, actions on A, B and C would be carried out simultaneously. However, if only one crew was available, the actions would be carried out based on the priority order defined in the template. Given that all aspects of maintenance can be probabilistically defined, the duration of a maintenance phase, unlike an operational phase, is not fixed and the phase lasts as long as it takes to complete all actions specified in the phase. To illustrate this, consider a race car that competes in two races, and even though corrective repair actions can be done during each race as needed, the race car then undergoes a major overhaul (i.e., series of maintenance actions). For this example assume the major sub-systems of the car undergoing these maintenance tasks are the engine, the transmission, the suspension system and the tires. The operation of the race car can then be represented as a phase diagram consisting of two operational phases, representing the two races, and one maintenance phase representing the maintenance activities. The figure below shows such a phase diagram along with the maintenance template.



Maintenance Phase Properties

Maintenance Template

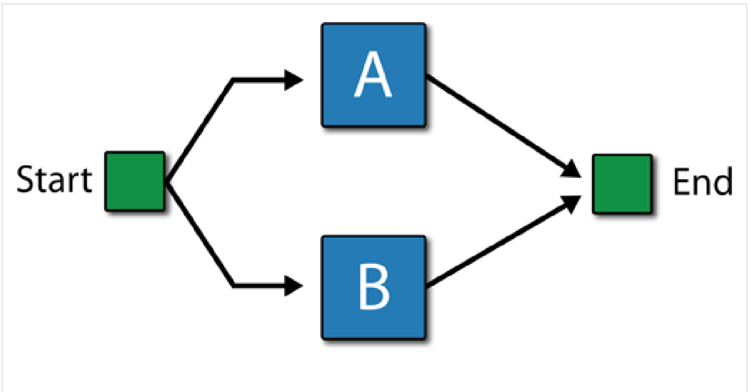
This specifies the maintenance template to be used in the currently selected maintenance phase.

Interval Maintenance Threshold

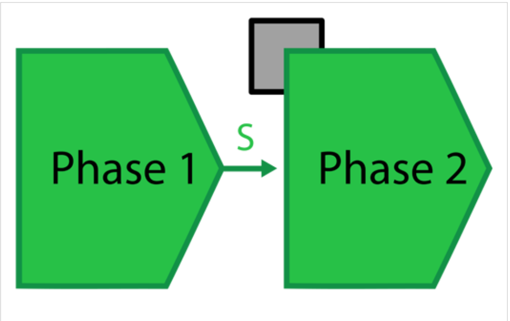
The Interval Maintenance Threshold property provides the ability to add some flexibility to the timing of scheduled maintenance tasks. In other words, tasks based on system or item age intervals (fixed or dynamic) will be performed if the start of the maintenance phase is within (1-X)% of the scheduled time for the action. It is used to specify an age interval when a maintenance task will be performed. This helps in optimizing the resources allocated to repair the system during a maintenance phase by performing preventive maintenance actions or inspections when the system is already down in a maintenance phase. For example, a preventive maintenance action is scheduled for a car (e.g., an oil change, tire rotation, etc.) every 60,000 miles, but a system downing failure of an unrelated component occurs at 55,000 miles. Here the system age threshold will determine whether the preventive maintenance will be performed earlier than scheduled. If the Interval Maintenance Threshold is 0.9, the preventive maintenance will be performed since the failure occurred after the system accumulated 91.67% of the time to the scheduled maintenance or is within 8.33%, (60,000-55,000)/60,000= 8.33%, of the system age at which the preventive maintenance was originally scheduled. If the system age threshold was 0.95, the preventive maintenance will not be performed at 55,000 miles, since the system failure did not occur within 5% of the system age at which the preventive maintenance was originally scheduled (1-0.95=0.05 or 5%).

For example, consider a system that has two components: Block A and Block B. The system undertakes a mission that can be divided into two phases. The first phase is an operational phase with a duration of 1,370 hours, with both the components in a parallel configuration. In this phase, Block A fails every 750 hours while Block B fails every 1,300 hours. Corrective maintenance on Block A in this phase requires 100 hours to be completed. A preventive maintenance of 20 hours duration also occurs on Block A every 500 hours. No maintenance can be carried out on Block B in this phase.

The second phase of the mission is a maintenance phase. In this phase, Block A has the same maintenance actions as those in the first phase. A corrective maintenance of 100 hours duration is defined for Block B. Phase 2 also has a value of 0.70 set for the Interval Maintenance Threshold. All maintenance actions during the entire mission of the system have a type II restoration factor of 1.



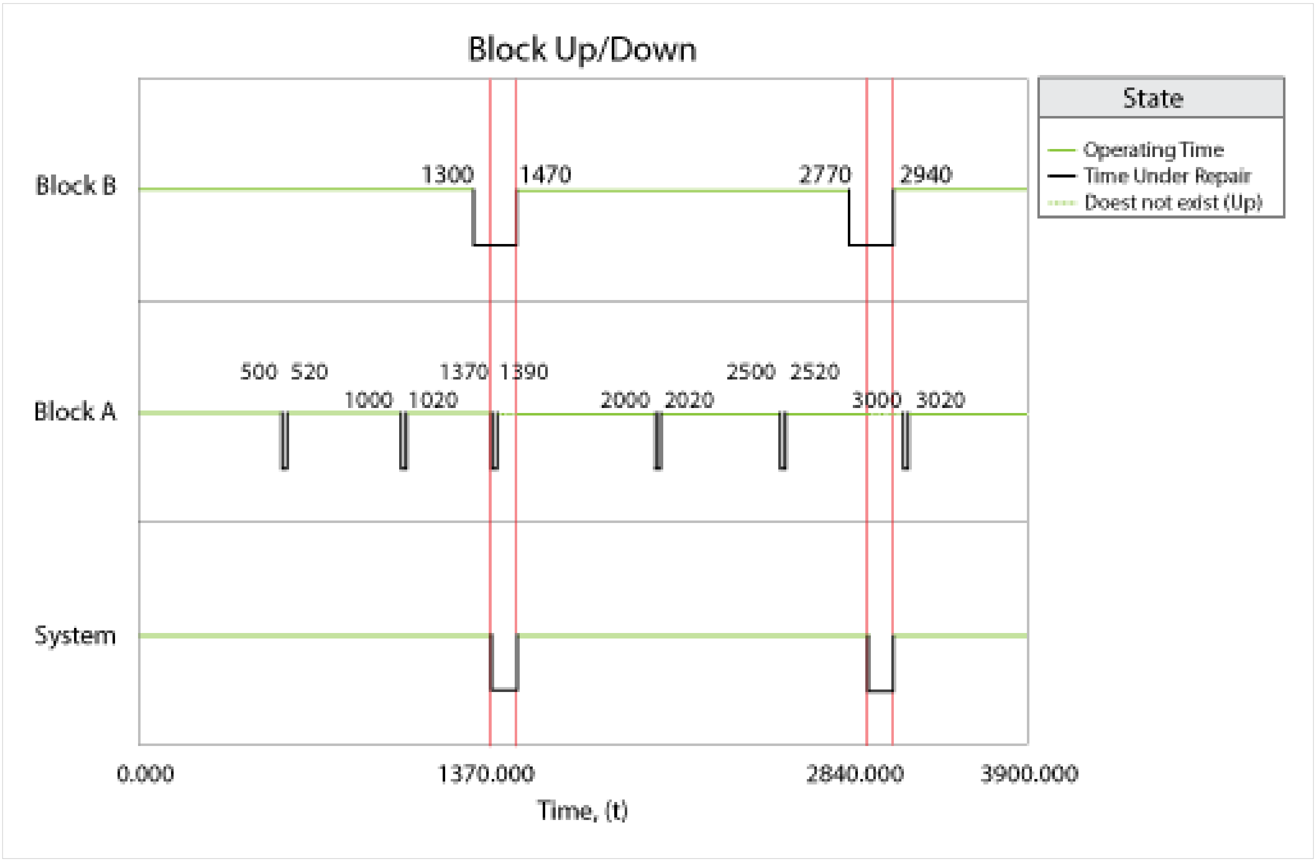
RBD for Phase 1



Phase Diagram

Name	Type	Duration	Block A				Block B			
			Failure	CM Duration	PM Interval	PM Duration	Failure	CM Duration	PM Interval	PM Duration
Phase 1	Operational	1370	750	100	500	20	1300	None	None	None
Phase 2	Maintenance	N/A	N/A	100	500	20	N/A	100	None	None

The system behavior from 0 to 3,500 hours is shown in the plot below and described next.



1. Phase 1 begins at time 0 hours. The duration of this phase is 1,370 hours.
2. At 500 hours, the first of the scheduled PMs for Block A begins. The duration of these maintenance tasks is 20 hours. The scheduled maintenance ends at 520 hours.
3. At 1,000 hours, another PM occurs for Block A based on the set policy. This maintenance ends at 1,020 hours.
4. At 1,300 hours, Block B fails after accumulating an age of 1,300 hours. A system failure does not occur because Block B is in a parallel configuration with Block A in this phase. Repairs for Block B are not defined in this phase. As a result Block B remains in a failed state.
5. At 1,370 hours, phase 1 ends and phase 2 begins. Phase 2 is a maintenance phase. Block B is repairable in this phase and has a CM duration of 100 hours. As a result, repairs on Block B begin and are completed at 1,470 hours. Block A has the next PM scheduled to occur at 1,500 hours. However, phase 2 has an Interval Maintenance Threshold of 0.7 for Preventive and Inspection Policies. The time remaining to the next PM is 130 hours (1,500-1,370 = 130 hours). This remaining time over the PM policy time of 500 hours is 26%. This is within 0.3 (1-0.70 = 0.3) or 30% corresponding to the threshold value of 0.70. Thus the PM task that is to occur at 1,500 hours is carried out in the maintenance phase from 1,370 hours to 1,390 hours, while no PM occurs at 1,500 hours. All maintenance actions are completed by 1,470 hours and phase 2 ends at this time. This completes the first cycle of operation for the phase diagram.
6. At 1,470 hours, phase 1 begins in the second cycle.
7. At 2,000 hours, the next PM for Block A begins. This maintenance ends at 2,020 hours.

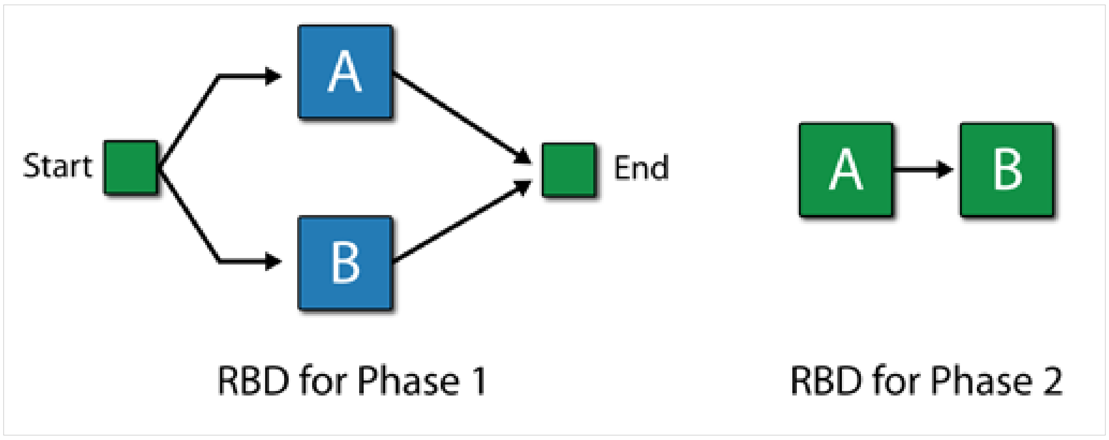
8. At 2,500 hours, another PM is carried out on Block A and is completed by 2,520 hours.
9. At 2,770 hours, Block B fails in the second cycle of phase 1 after accumulating an age of 1,300 hours. Since no repair is defined for the block in this phase, it remains in a failed state.
10. At 2,840 hours, phase 1 completes its duration of 1,370 hours and ends. Phase 2 begins in the second cycle and the corrective maintenance, defined for a duration of 100 hours for Block B, begins. This repair action ends at 2,940 hours. For Block A, the time remaining until the next PM at 3,000 hours is 160 hours (3000-2840 = 160 hours). This remaining time over the PM policy of 500 hours is 32%. This is not within 30% corresponding to the threshold value of 0.70. Thus the PM due at 3,000 hours is not considered close enough to the beginning of the maintenance phase and is not carried out in this phase. At 2,940 hours, all maintenance actions are completed in phase 2 and phase 2 ends. This also completes the second cycle of operation for the phase diagram.
11. At 2,940 hours, phase 1 begins in the third cycle.
12. At 3,000 hours, the scheduled maintenance on Block A occurs. This PM ends at 3,020 hours.
13. At 3,500 hours, the simulation ends.

Success/Failure Paths

For a failure path, on system failure, the system goes to somewhere immediately when the failure occurs. For a success path, if there is no system failure during this phase, the system goes to somewhere by the end of the current phase.

In BlockSim 7, there was only one path from each phase. The failure outcome for each phase was defined via a drop-down list. The success and failure paths used in BlockSim 8 and above make it easy to see what will happen upon success or failure for each block. They also allow you to create more complex phase diagrams, in which success and failure may lead to entirely different sets of phases. In addition, they offer an additional possible outcome of failure. Previously, the outcome of failure could be maintenance, stopping the simulation or continuing the simulation; now another possible outcome of failure can be continuing simulation on a different path.

For example, consider a system whose RBDs for Phase 1 and Phase 2 are as shown in the figure below:

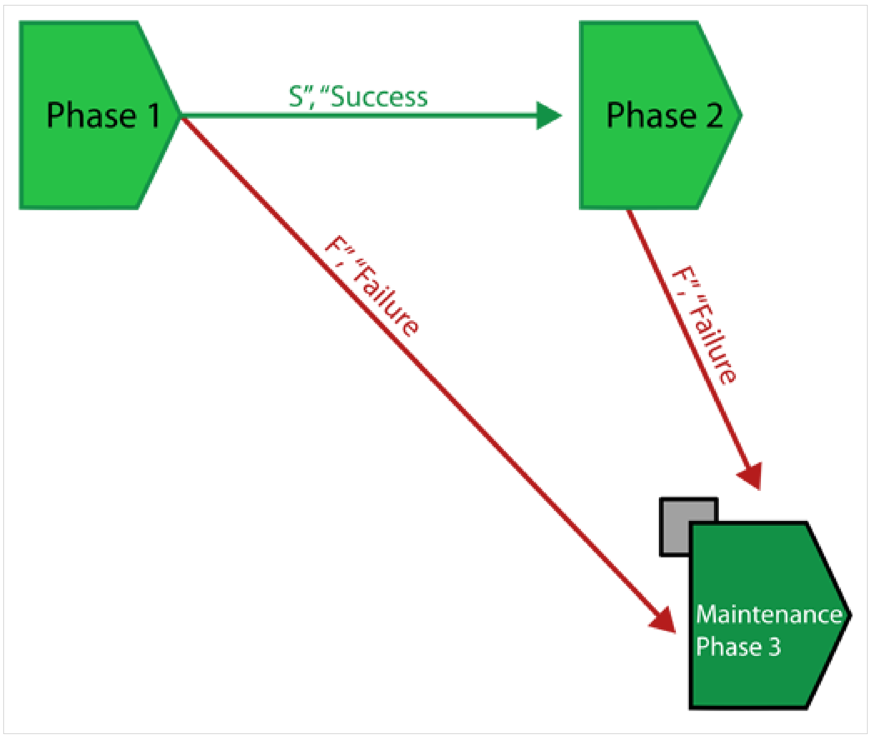


In Phase 1, Block A follows Weibull distribution with Beta = 1.5 and Eta = 1000 hours. Block B follows Weibull distribution with Beta = 1.5 and Eta = 1200 hours. No maintenance.

In Phase 2, Block A follows Weibull distribution with Beta = 1.5 and Eta = 900 hours. Block B follows Weibull distribution with Beta = 1.5 and Eta = 1150 hours. No maintenance.

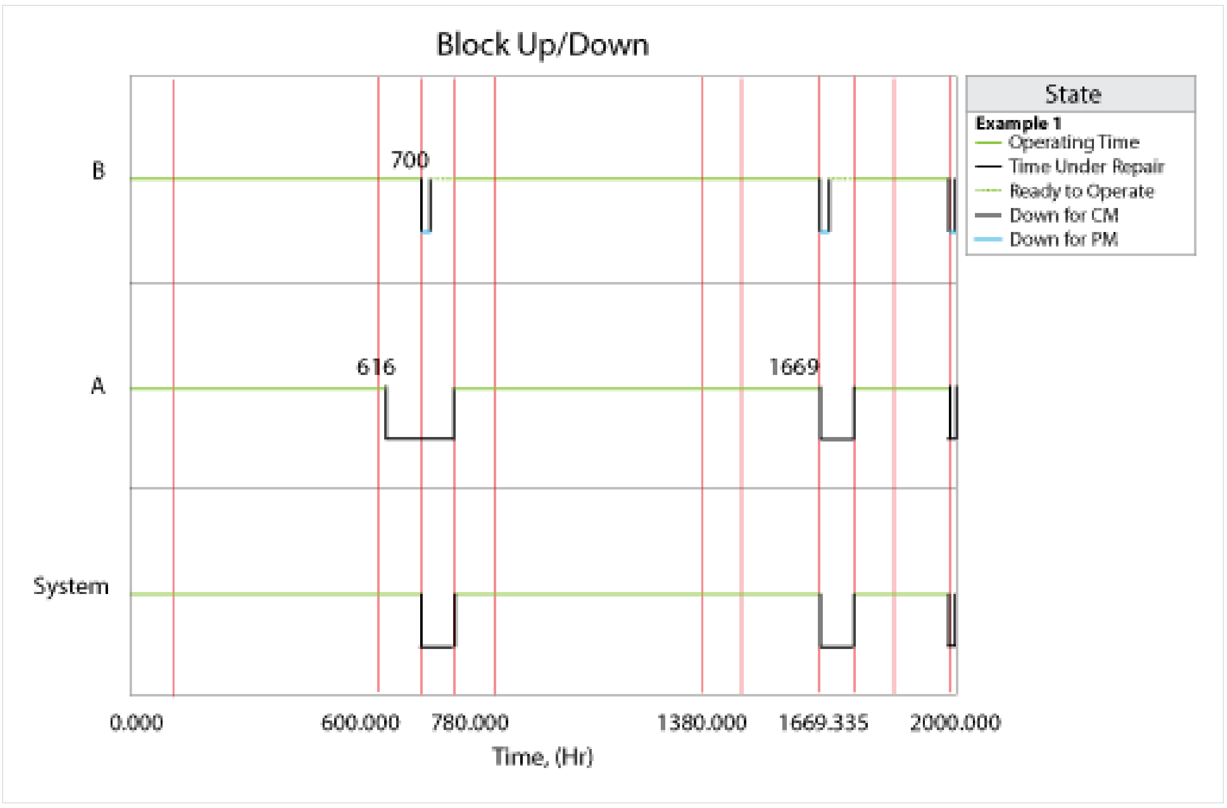
Scenario 1

Suppose the phase diagram is as shown next. The maintenance phase has a CM with duration 80 hours and PM with duration 20 hours for both Block A and B. Phase 1 has a duration 100 hours and Phase 2 has a duration 500 hours. If there is no failure in Phase 1, upon finish, it goes to Phase 2. If there is a failure in Phase 1, it would immediately go to Maintenance Phase 3. If there is no failure in Phase 2, upon finish, it goes back to Phase 1. If there is a failure in Phase 2, it would immediately go to Maintenance Phase 3.



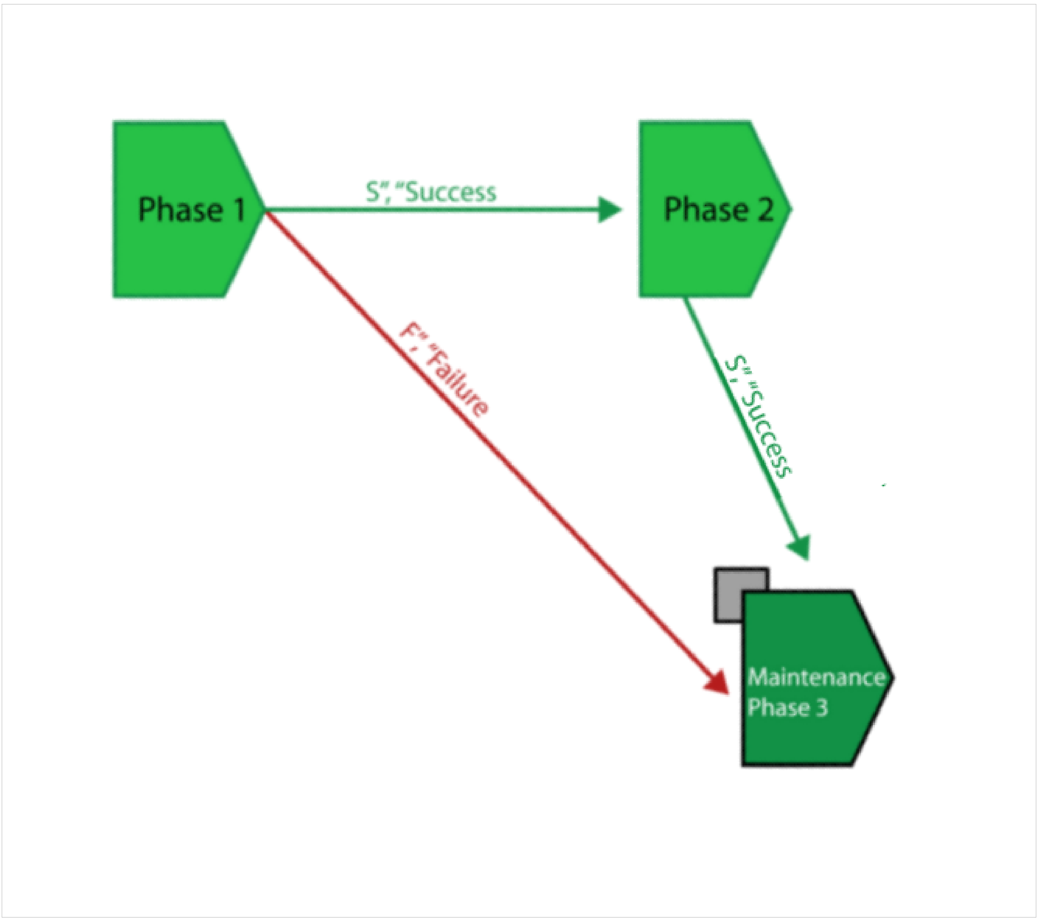
The Block Up/Down plot is shown below:

1. At the first cycle (0-100 hours for Phase 1 and 200-600 hours for Phase 2), there is no failure, thus after Phase 2, Phase 1 is executed (back to Phase 1 after this cycle).
2. At 616 hours (Phase 1 of the second cycle), Block A fails. The system doesn't enter Maintenance Phase 3 immediately because in Phase 1, Block A and Block B are parallel. Failure of only Block A does not bring the system down.
3. At 700 hours, after Phase 1, the system goes into Phase 2. In Phase 2, the failure of Block A (passed from Phase 1) brings the system down because in Phase 2, Block A is in series with Block B. Thus the system goes into Maintenance Phase 3 immediately. Block B is not down, thus it gets PM with duration 20 hours. Block A gets CM with duration 100 hours.
4. At 1669 hours, Block A fails in Phase 2, which brings down the system and the system goes into Maintenance Phase 3 immediately. Block A gets CM and Block B gets PM.


Block Up/Down
B
700
A
616
1669
System
0.000
600.000
780.000
1380.000
1669.335
2000.000
Time, (Hr)
State
Example 1
Operating Time
Time Under Repair
Ready to Operate
Down for CM
Down for PM

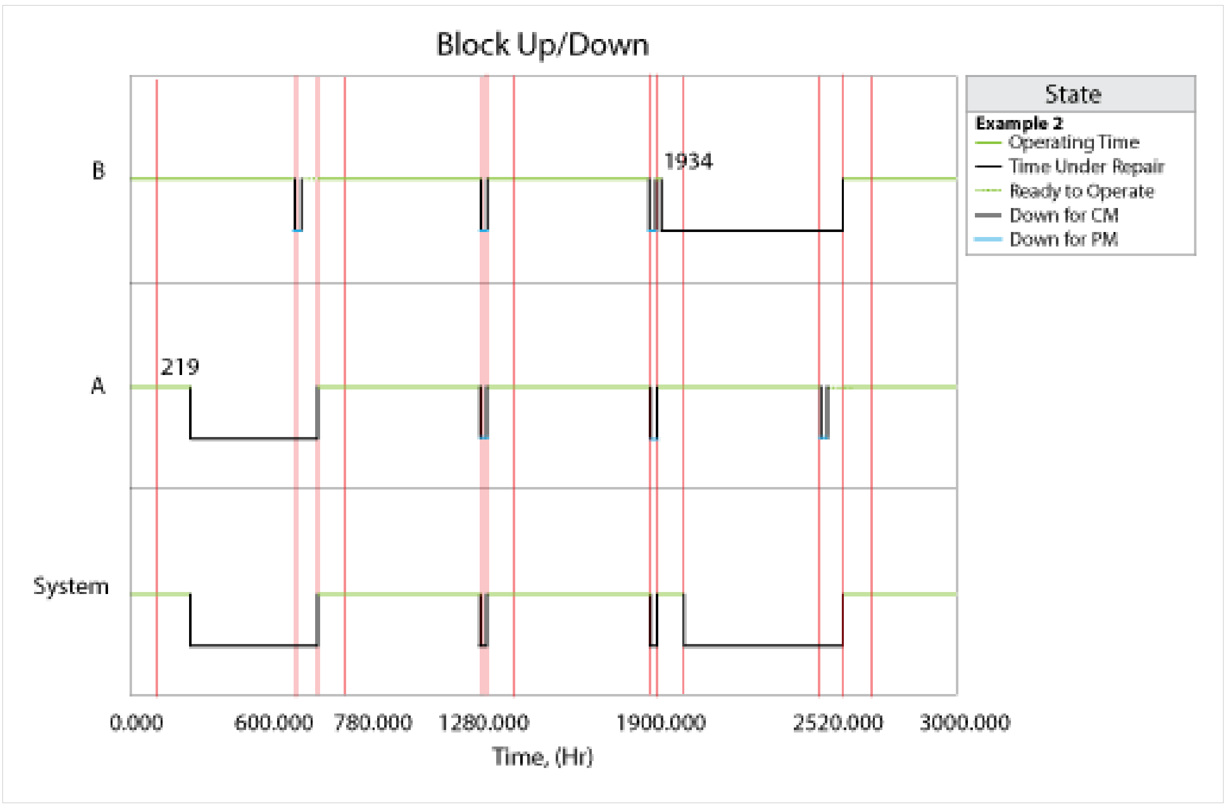
Scenario 2

Now suppose that the phase diagram is as follows, where everything is the same as in Example 1 except that the path from Phase 2 to Maintenance Phase 3 changes from a failure path to a success path. This means that whether or not there is a failure in Phase 2, the system would go to the Maintenance Phase 3 after Phase 2. But if there is a failure in Phase 2, the system would go to the Maintenance Phase 3 after Phase 2, not immediately.



The Block Up/Down plot is shown below:

1. At 219 hours, Block A fails in Phase 2. This failure brings the system down. However, since the path from Phase 2 to Maintenance Phase 3 is a success path, the system doesn't go into Maintenance Phase 3 immediately. Instead, it goes into Maintenance Phase 3 after Phase 2 ends. Block A gets CM and Block B gets PM in the maintenance phase.
2. At 1934 hours, Block B fails in Phase 1. This failure doesn't bring the system down because in Phase 1, the system is in parallel structure. After Phase 1, the system goes into Phase 2 and the failure of Block B brings they system down in Phase 2. However, the system doesn't go into Maintenance Phase 3 immediately because the path between them is a success path. The system goes into Maintenance Phase 3 after Phase 2 ends, and Block B gets CM and Block A gets PM there.



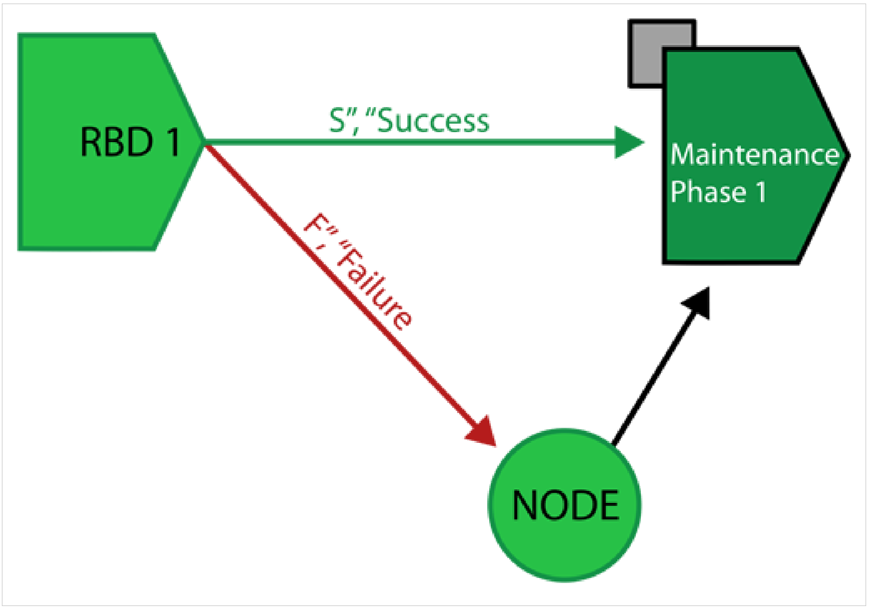
Node Blocks and Stop Blocks

Starting with Version 8, the two possible outcomes of an operational phase block are modeled using success and failure paths. Where previously a failure outcome was defined as part of the operational phase block's properties, it is now graphically represented within the diagram. Node blocks and stop blocks are provided to allow you to build configurations that are both accurate and readable.

Node Blocks

The purpose of a node block is simply to enable configurations that would otherwise not be possible due to limitations on connecting blocks. For example, consider an instance where maintenance is scheduled to be performed after the operational phase has completed successfully, and if a failure occurs during simulation, that maintenance will take place upon failure. In this case, the operational phase block's success and failure outcomes are identical. Success paths and failure paths cannot be identical in phase diagrams, however, so you would model this configuration in one of two ways:

- If the operational phase stops upon failure of the block and the simulation moves to the next phase along the success path, you would use a node block to model this configuration, as shown next.

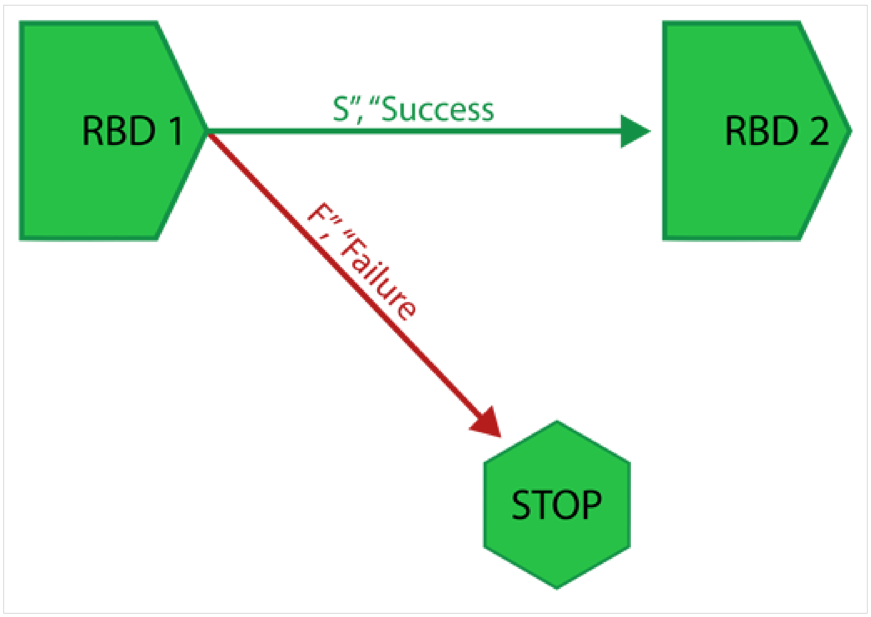


- If the operational phase continues for the specified duration despite failure and the simulation then moves to the next phase along the success path, you would simply not create a failure path.
  - If there is only one path, the success path observed for a phase, then on system failure at this phase, the "continue simulation" rule of BlockSim 7 applies. Under "continue simulation," when a system failure occurs, repairs begin as per the repair policy selected and the time to restore the system is part of the operational phase's time. In other words, the repairs continue in the operational phase until the system is up again. If the repairs are not completed before the phase ends, the repairs continue into the next phase. Thus, under this rule the duration of an operational phase is not affected by a system failure. As an example of this rule, consider a production line operating in two phases: a day shift and a night shift. A failure occurs in the day shift that renders the production line non-operational. Repair of the production line begins immediately and continues beyond the day shift. The production line is back up after midnight. In this case, the repair of the production line exhausts all of the duration of the day shift phase from the time of the failure to the end of the phase. Some part of the night shift phase is also exhausted.

Node blocks can have unlimited incoming connections and a single outgoing connection.

Stop Blocks

Stop blocks indicate that the simulation of the mission ends. A new simulation may then begin, if applicable. This is useful in situations where maintenance is not possible upon failure.



Stop blocks can have unlimited incoming connections. No outgoing connections can be defined for stop blocks.

When a path leads to a stop node, it is the same as the option "Start New Simulation" in BlockSim 7, which would halt the simulation and effectively means the end of the mission if the system fails. Specifically, if a failure path leads to a stop node the execution of the current operational phase and all phases that follow the current phase is halted, and the mission aborted. The stop node can be used to model a system whose failure cannot be repaired and the mission has to be aborted if a failure occurs. A good example of this would be the aircraft case discussed previously. A catastrophic failure during cruising would end the mission.

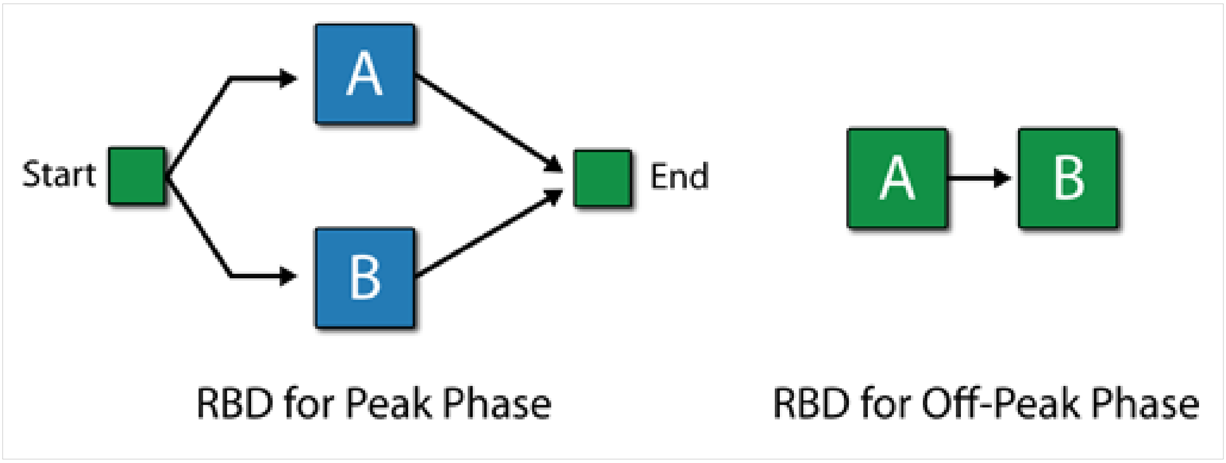
Subdiagram Phase Blocks

Subdiagram phase blocks represent other phase diagrams within the project. Using subdiagram phase blocks allows you to incorporate phase diagrams as phases within other phase diagrams. This allows you to break down extremely complex configurations into smaller diagrams, increasing understandability and ease of use and avoiding unnecessary repetition of elements.

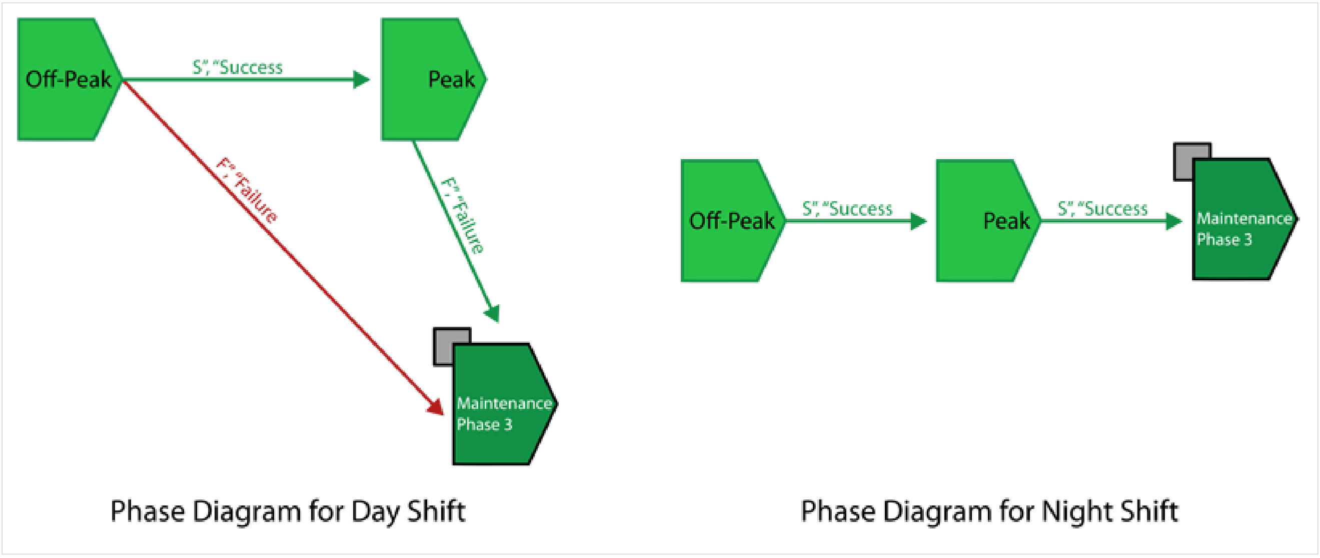
Subdiagram phase blocks can have unlimited incoming connections and up to two outgoing connections, which may include one success path and one failure path. The success path and the failure path must be different; if both success and failure of the block actually lead to the same outcome, you can use a node block to model this configuration.

To illustrate how subdiagram blocks are handled in BlockSim's simulations, consider a server that has two shifts each day, a day shift lasting 8 hours and a night shift lasting 16 hours. Within each shift, it has a peak period and an off-peak period with different durations.

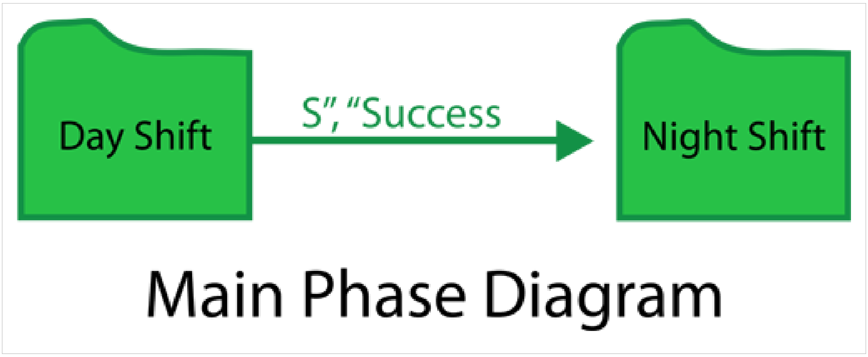
The RBDs for the Peak phase and the Off-Peak phase are:



The phase diagrams for the Night Shift and Day Shift are:



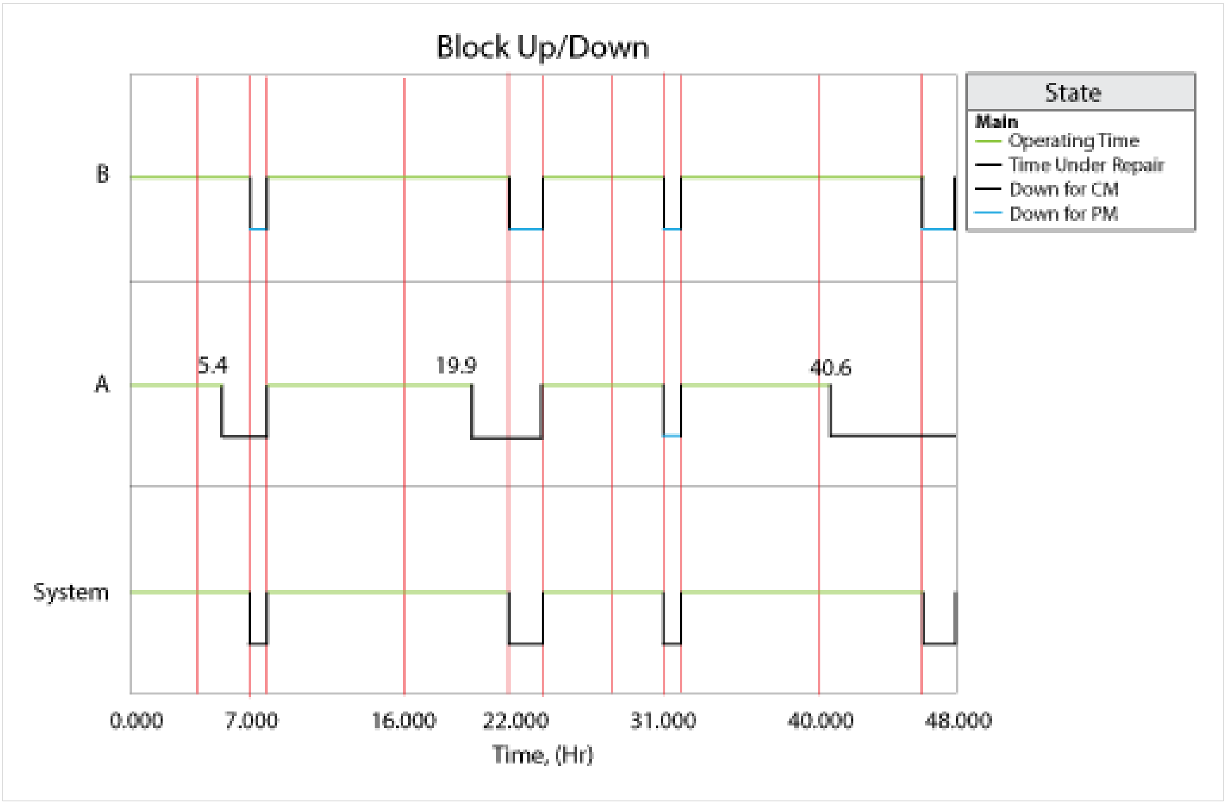
The main phase diagram with phase subdiagram blocks is:



During the Peak phase, the reliability of Block A follows W(1.5,30) and the reliability of Block B follows W(1.5,40). During the Off-Peak phase, the reliability of Block A follows W(1.5,15) and the reliability of Block B follows W(1.5,20). In the Day Shift diagram, both blocks have CM and PM with duration = 1 hour; in the Night Shift diagram, both blocks have CM and PM with duration = 2 hours. In the Day Shift diagram, the duration of the Off-Peak phase is 4 hours and the duration of the Peak phase is 3 hours. In the Night Shift diagram, the duration of the Off-Peak phase is 8 hours and the duration of the Peak phase is 6 hours.

The system log for 2 days (48 hours) is shown in the figure below.

1. At 5.4 hours, in the Peak phase of the Day Shift diagram, Block A fails. After this Peak phase, Block A gets CM and Block B gets PM in the maintenance phase.
2. At 19.9 hours, in the Peak phase of the Night Shift diagram, Block A fails again. After this Peak phase, Block A gets CM and Block B gets PM in the maintenance phase.
3. At 40.6 hours, in the Peak phase of the Night Shift diagram, Block A fails again. After this Peak phase, Block A gets CM and Block B gets PM in the maintenance phase.



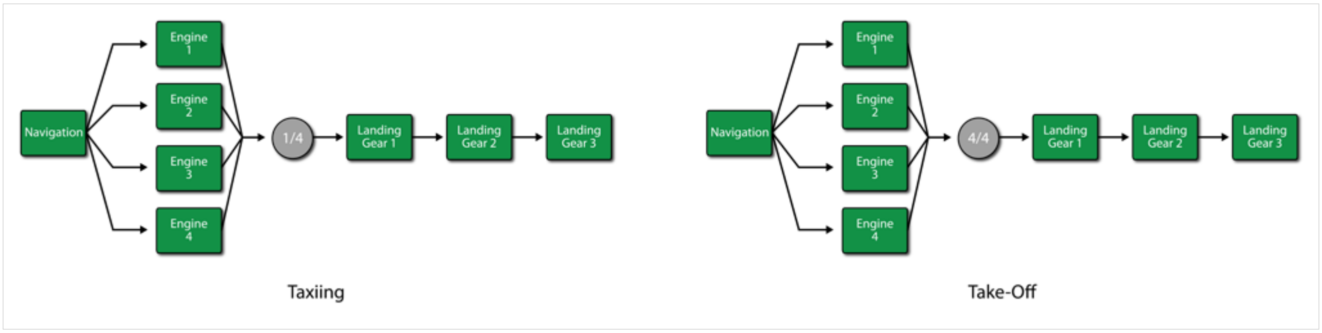
Example: Aircraft Phases with Forced Landing

Consider simulating a flight mission for a four-engine aircraft. The configuration of the aircraft changes at different time. During the taxiing phase, the navigation system, one out of the four engines and all three landing gears are needed. During the take-off phase, all four engines, the navigation system and all three landing gears are needed. During the cruising phase, the navigation system and three of the four engines are required. During the landing phase, the navigation system, two of the four engines and all three landing gears are required. After finishing a journey with these four phases, the aircraft goes into a maintenance phase for maintenance. However, in addition to the four operational phases discussed, there is a special operational phase called the forced landing phase. This phase is used for modeling the case when there is a failure during take-off, cruising or landing phase, forcing the aircraft to land. If the forced landing succeeds, the aircraft goes into the maintenance phase. However, if the forced landing fails, the aircraft crashes and the simulation ends.

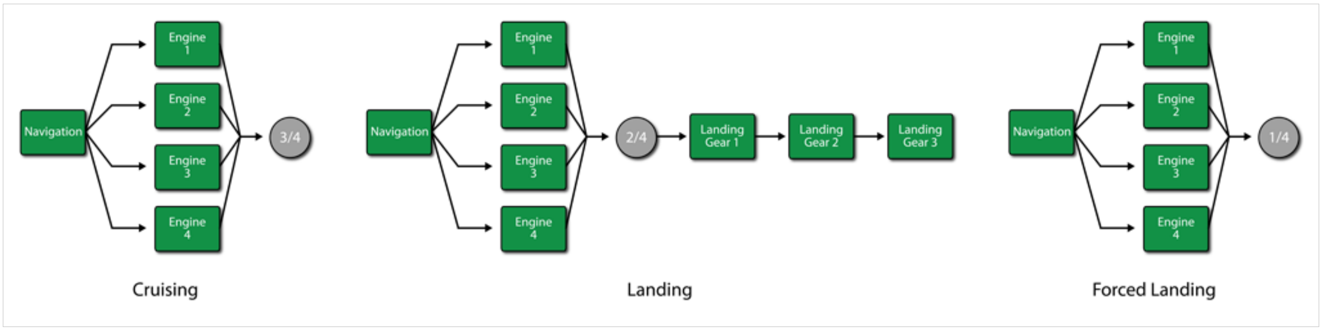
The purpose of this example is to illustrate how to create a nonlinear system with Failure/Success links in phase simulation.

RBDs and Phase Diagram

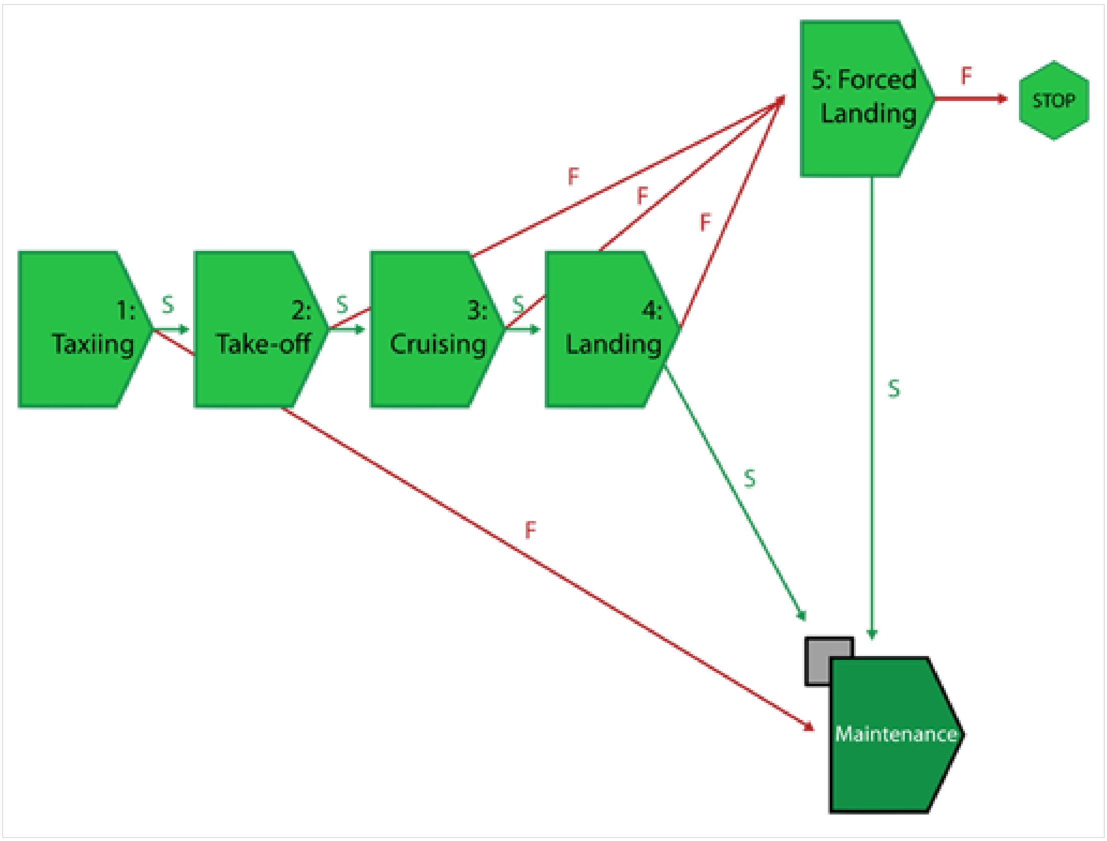
The RBDs for the Taxiing phase and the Take-Off phase are:



The RBDs for other three phases are:



The Phase diagram is:

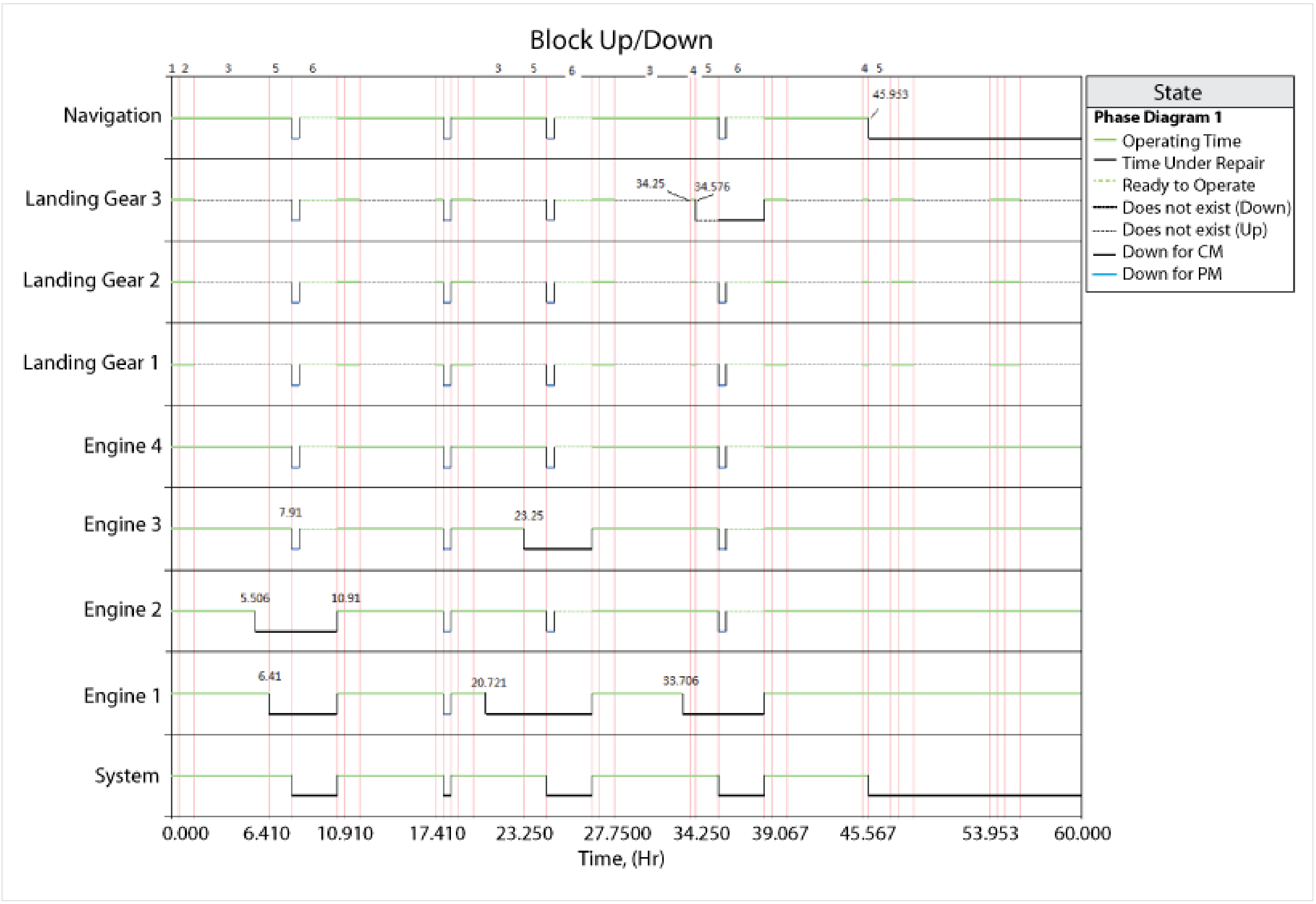


The reliability of the navigation system follows a Weibull distribution with Beta=1.5 and Eta=30 hours. The reliability of the engines follows a Weibull distribution with Beta = 1.5 and Eta = 20 hours. The reliability of the landing gears follows a Weibull distribution with Beta = 1.5 and Eta = 15 hours.

In the maintenance phase, all blocks have CM with duration = 3 hours. CM is performed upon item failure. All blocks also are subjected to PM with duration = 0.5 hours and the PM is performed when the maintenance phase starts. Both CM and PM restore the blocks to as good as new.

Block Up/Down plot

1. At time 5.506 hours, in the Cruising phase, Engine 2 fails. However, this failure doesn't bring the system down because in the Cruising phase, only three of the four engines are required.
2. At time 6.41 hours, in the Cruising phase, Engine 1 fails too. This failure brings the system down and the system goes into the Forced Landing phase immediately.
3. At time 7.91 hours, the Forced Landing phase is done. No failure happened in this phase, which means the forced landing is successful. The system goes into the Maintenance phase. All blocks except Engine 1 and Engine 2 get PM in the Maintenance phase.
4. At time 10.91 hours, Engine 1 and Engine 2 are done with their CM and the system goes back to the Taxiing phase.
5. At time 20.721 hours, in the Cruising phase, Engine 1 fails again. Of course, this failure doesn't bring the system down.
6. At time 23.25 hours, in the Cruising phase, Engine 3 fails and this failure brings the system down. The system goes into the Forced Landing phase. After the Forced Landing phase, the system goes into the Maintenance phase.
7. At time 33.706 hours, in the Cruising phase, Engine 1 fails again. This failure doesn't bring the system down.
8. At time 34.25 hours, the system goes into the Landing phase.
9. At time 34.576 hours, in the Landing phase, Landing Gear 3 fails and this failure brings the system down. The system goes into the Forced Landing phase immediately.
10. At time 45.953 hours, in the Landing phase, the navigation system fails. This failure brings the system down and the system goes into the Forced Landing phase immediately. In the Forced Landing phase, the navigation system is in series with other blocks, the failure of the navigation system brings the system down. Thus there is a system failure in the Forced Landing phase, which means the forced landing fails and the system goes into the Stop node. Simulation stops.



Simulation Rules and Assumptions

When Transferring Interrupted Maintenance Tasks Across Phases

Maintenance tasks in progress during one operational phase can be interrupted if that phase ends before the repair is completed. For example, a crew delay or spare parts order may extend the duration of a repair beyond the duration of the phase. As described next, the software handles these interruptions differently, based on the stage in which the repair was interrupted and whether or not the failed block is present in the next contiguous phase.

1. If a phase ends during the repair of a failed block and the block is present in the next contiguous phase:

    a) If the same task is present in both phases, then the task will continue as-is in the next phase. This is considered an uninterrupted event, and counts as a single unique event at both the block and the system level.

    b) If the interrupted task is not used in the next phase, then the task is cancelled and new tasks are applied as needed. In this case, all crew calls are cancelled and spare parts are restocked.

        1) If the repair has started or the crew is delayed (crew logistic delay), the call will be assumed accepted and the component will be charged for it. If the crew was occupied with another component's repair, the call will be assumed rejected and hence not charged to the component.

        2) If the call for spare parts incurred emergency charges, those are charged to the block; otherwise, there are no other charges to the block.

2. If a phase ends during the repair of a failed block and the block is not present in the next contiguous phase, then the task is cancelled and new tasks are applied as needed. All crew calls are cancelled and spare parts are restocked:

   a) If the repair has started or the crew is delayed (crew logistic delay), the call will be assumed accepted and the component will be charged for it. If the crew was occupied with another component's repair, the call will be assumed rejected and hence not charged to the component.

   b) If the call for spare parts incurred emergency charges, those are charged to the block; otherwise, there are no other charges to the block.

   c) Discontinuous events are counted as two distinct events at both the block and the system level.

   d) When the system fails in a phase that has a failure path leading to a stop block, the system will remain down for the remainder of the simulation. From that point on, the blocks that are down are assumed unavailable and the blocks that are up are assumed operational for availability calculations.

For Stop Blocks

When a system failure occurs in a phase where the failure path points to a stop block, the simulation is aborted. Once this failure occurs, the following assumptions apply to the results:

• Components that are under repair or maintenance remain down and unavailable for the rest of the simulation.

• Components that are operating remain up for the rest of the simulation.

For a Maintenance Phase

A system is considered down and unavailable during the execution of a maintenance phase and remains down until all components have been repaired or maintained according to the properties specified for the maintenance phase. A maintenance phase is executed when the simulation reaches the phase while progressing through the phase diagram, either following a success path or a failure path. The following assumptions apply to both cases.

1. When a component enters a maintenance phase in a down state, the following rules apply:

   a) If a task is in progress for this component, the event will transfer to the maintenance phase provided that the same task is present in the maintenance phase. The rules for interrupted tasks apply as noted above.

   b) If the component is failed but no corrective maintenance is in progress (either because the component was non-repairable in the phase where it failed or because it had a task scheduled to be executed upon inspection and was waiting for an inspection), a repair is initiated according to the corrective maintenance properties specified for the component in the maintenance phase.

   c) Failed components are fixed in the order in which they failed.

2. When a component enters a maintenance phase in an operating state, the following rules apply:

   a) Maintenance will be scheduled as follows:

      1) Tasks based on intervals or upon start of a maintenance phase

      2) Tasks based on events in a maintenance group, where the triggering event applies to a block

      3) Tasks based on system down

      4) Tasks based on events in a maintenance group, where the triggering event applies to a subdiagram

Within these categories, order is determined according to the priorities specified in the maintenance template (i.e., the higher the task is on the list, the higher the priority).

b) An inspection or preventive task may be initiated, if applicable, with inspections taking precedence over preventive tasks. Inspections and/or preventive tasks are initiated if one of the following applies:

1) Upon certain events:

a) The task is set to be performed when a maintenance phase starts.

b) The policy is set to be performed based on events in a maintenance group and one of those events occurs within the one of the specified maintenance groups. Note that such a triggered maintenance does not follow the priorities specified in the maintenance template, but is sent to the end of the queue for repair.

c) The task is set to be performed whenever the system is down.

2) At certain intervals:

a) The task is set to be performed at a fixed time interval, based on either item age or calendar time, and the maintenance falls within the maintenance threshold specified in the maintenance phase.

If the inspection task is not set to bring either the item or the system down, the inspection will still be considered a downing inspection.

Finally, if a block enters a maintenance phase in a failed state:

1. If the block does not have a corrective task in the maintenance phase but does have an on condition task, the preventive portion of the on condition task is triggered immediately in order to restore the block.
2. A maintenance phase will not end until all components are restored. Therefore, if any failed block does not have a task that restores it, the maintenance phase will not end.

Phase Throughput

Phase throughput is the maximum number of items that a system can process during a particular phase. It is defined at the phase level as a phase property in an operational phase. For a detailed discussion of throughput at the block level see Throughput Analysis. Phase throughput can be thought of as the initial throughput that enters the system. For example, imagine a textile factory that receives different quantities of raw materials during different seasons. These seasons could be treated as different phases. In this case a phase may be seen as sending a certain quantity of units to the first component of the system (the textile factory in this case). Depending on the capacity and availability of the factory, these units may be all processed or a backlog may accumulate.

Alternatively, phase throughput can be used as a constraint to the throughput of the system. An example would be the start up period in a processing plant. When the plant stops operating, the equipment requires a warm up period before reaching its maximum production capacity. In this case the phase throughput may be used to limit the capacity of the first component which in turn would limit the throughput of the rest of the system. Note that there is no phase-related backlog for this example. In BlockSim this can be modeled by checking the **Ignore backlog** option in the Block Properties window for the first component.

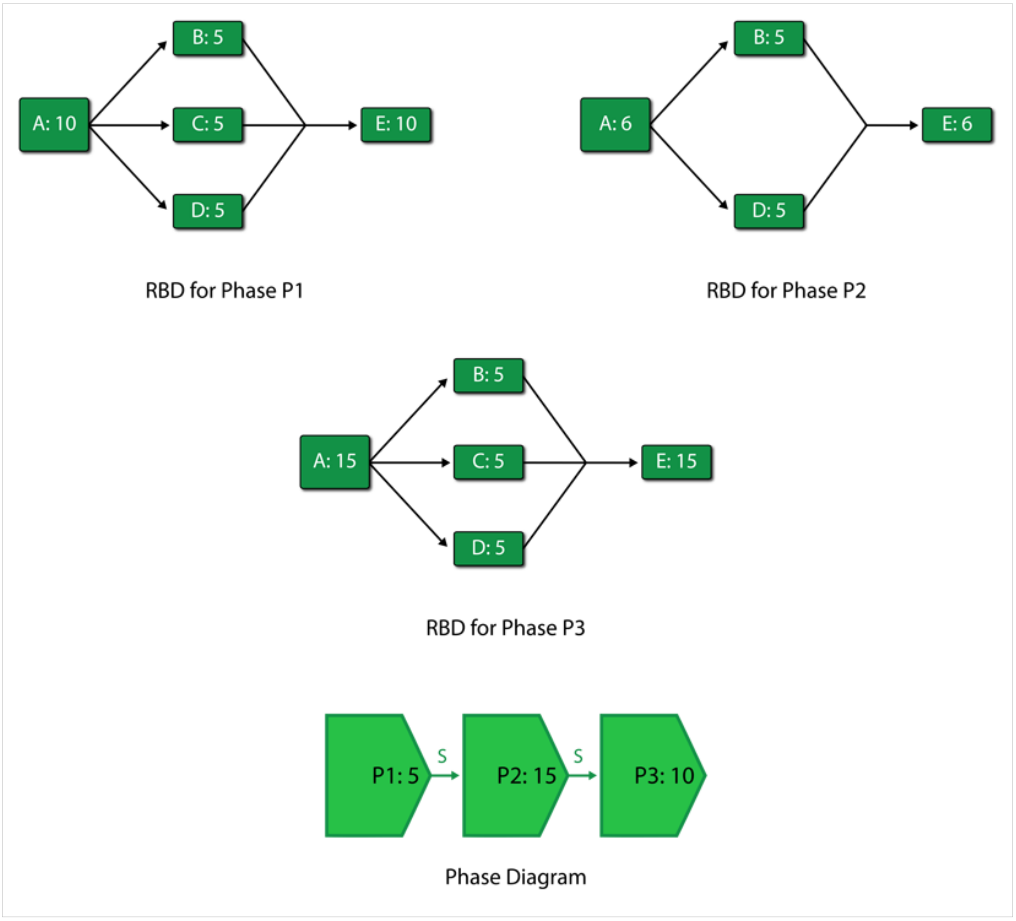
Phase throughput can be one of the following:

- **Unrestricted throughput** specifies that the amount of output the system can process during the phase will be equal to the amount that the linked diagram can process during the specified phase duration.
- **Constant throughput** allows you to specify an amount of throughput per unit time that is the same throughout the entire simulation.

- **Variable throughput** allows you to specify an amount of throughput per unit time that varies over the course of the simulation. For example, the flow of oil from a well may drop over time as the oil reserves are depleted so the amount of throughput per unit time will decrease as the simulation time increases.

Constant Throughput Example

To examine throughput in phases, consider a phase diagram of three phases with linked RBDs, as shown in the figure below. Each phase has a duration of 10 hours and all phases are set to continue simulation upon failure. Blocks A through E process a number of items per hour as identified next to each letter (e.g., A:10 implies that 10 units are processed per hour by block A). Similarly, each phase is marked with the constant throughput sent to the blocks in the phase (e.g., P1:5 implies phase P1 sends 5 units to its blocks). For the sake of simplicity, it is assumed that the blocks can never fail and that items are routed equally to each path (see Throughput Analysis Options). The ignore backlog option is left unchecked for all blocks in the phase diagram.



If the phase diagram is run on a throughput simulation with an end time of 60 hours, the following scenario can be observed:

1. Phase P1 from 0 to 10 hours

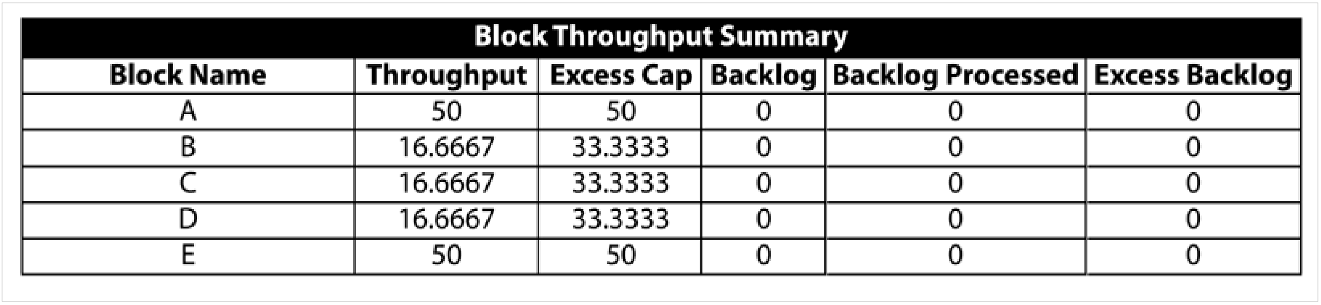
   a) Phase P1 sends 5 units in the first hour to block A. The capacity of block A in this phase is 10 units per hour. Block A processes these 5 units with an excess capacity of 5 units in the first hour.

   b) The 5 units processed by block A are routed equally over the three paths to blocks B, C and D. Each of these blocks receives 1.67 units in the first hour. The capacity of each of these blocks is 5 units per hour. Thus, blocks B, C and D each process 1.67 units with an excess capacity of 3.33 units in the first hour.

c) Blocks B, C and D route each of their 1.67 units to E. The capacity of E in this phase is 10 units per hour. E processes the total of 5 units with an excess capacity of 5 units in the first hour.

d) The above steps are repeated for the first ten hours to complete phase P1.

A summary result table for phase P1 is shown next.



Block Throughput Summary					
Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	50	50	0	0	0
B	16.6667	33.3333	0	0	0
C	16.6667	33.3333	0	0	0
D	16.6667	33.3333	0	0	0
E	50	50	0	0	0

2. Phase P2 from 10 to 20 hours

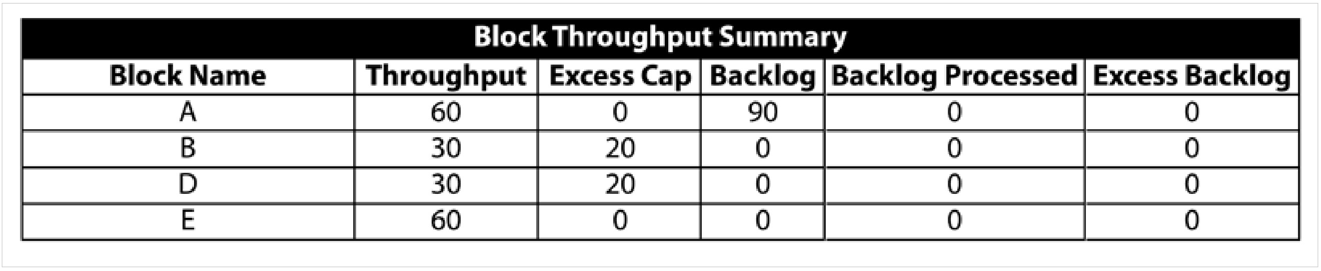
a) Phase P2 sends 15 units in its first hour to block A. The capacity of block A in this phase is changed from the previous phase and is now 6 units per hour. Thus, block A is able to process only 6 units while there is a backlog of 9 units per hour.

b) The 6 units processed by block A are routed equally to blocks B and D as C is absent in this phase. Blocks B and D get 3 units each. The capacity of each of these blocks is 5 units per hour. Thus, blocks B and D process 3 units with an excess capacity of 2 units each in the first hour of the second phase.

c) Blocks B and D route a total of 6 units to E. The capacity of block E in this phase is 6 units per hour. Thus, block E processes 6 units in the first hour of phase 2 with no excess capacity and no backlog.

d) The above steps are repeated for the ten hours of phase P2. At 20 hours phase P2 ends.

A summary result table for phase P2 is shown next.



Block Throughput Summary					
Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	60	0	90	0	0
B	30	20	0	0	0
D	30	20	0	0	0
E	60	0	0	0	0

3. Phase P3 from 20 to 30 hours

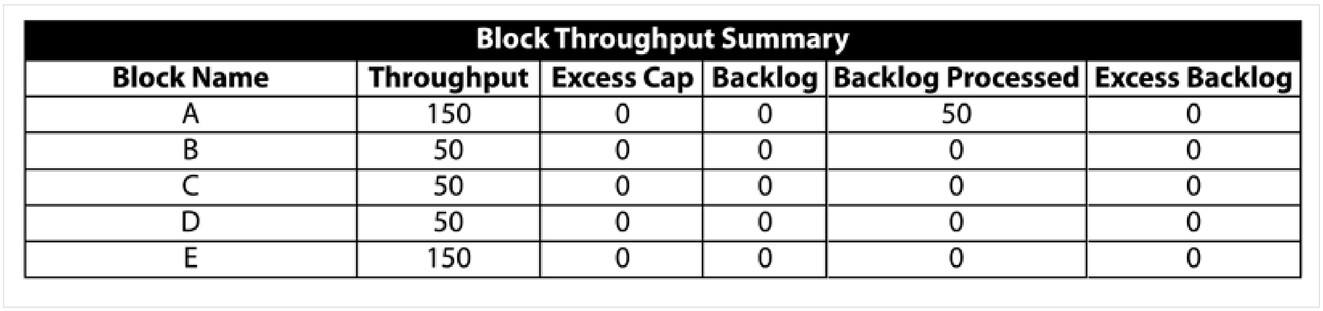
a) Phase P3 sends 10 units in its first hour to block A. The capacity of block A in this phase is now 15 units per hour. Thus, block A is able to process all 10 units sent by P3 and also process 5 units of the backlog from phase P2. Thus, during the first hour of phase P3, block A processes 15 units that includes a backlog of 5 units. Since block A is used to its full capacity, the excess capacity of A is zero.

b) The 15 units processed by block A in the first hour are routed equally over three paths to blocks B, C and D as block C is available again in this phase. Blocks B, C and D receive 5 units each. The capacity of each of these block is 5 units per hour. Thus, blocks B, C and D process 5 units each to full capacity with no excess capacity during the first hour of the third phase.

c) Blocks B, C and D route each of their 5 units to E. The capacity of block E in this phase is 15 units per hour. Thus, block E processes 15 units in the first hour of phase P3 with no excess capacity and no backlog.

d) The above steps are repeated for the ten hours of phase P3. It should be noted that although some of the backlog from phase P2 gets processed in phase P3 $(5 \times 10 = 50$units from the backlog get processed), not all 90 backlogged units are processed ($90 - 50 = 40$units remain). The remaining units cannot be shown as backlog for phase P3 because these were generated in phase P2. To avoid confusion between backlogs of different phases, BlockSim does not display backlog values at the individual phase levels. At 30 hours, phase P3 ends. This also ends the first cycle of the phase diagram.

A summary result table for phase P3 is shown next.



Block Throughput Summary					
Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	150	0	0	50	0
B	50	0	0	0	0
C	50	0	0	0	0
D	50	0	0	0	0
E	150	0	0	0	0

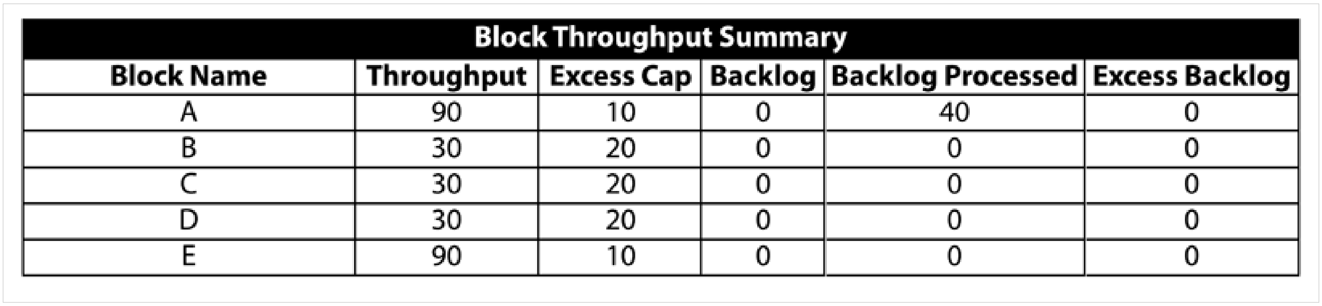
4. Phase P1 in the second cycle from 30 to 40 hours

a) As in the first cycle, phase P1 sends 5 units to block A in the first hour. The capacity of block A in this phase is 10 units per hour. Thus, block A is able to process all 5 units sent by P1 and also process 5 units of the backlog from the first cycle. Thus, during the first hour of the second cycle of phase P1, block A processes 10 units that includes a backlog of 5 units. Since there is a backlog of 40 units from the first cycle, this is processed by block A in the first eight hours of the second cycle. During the last two hours of the second cycle of phase P1, block A processes only the 5 units per hour it receives from P1 and has an excess capacity of 5 units per hour. Thus, by the end of this phase, block A processes a total of 90 units (i.e., 50 units from P1 and 40 units of backlog) with an excess capacity of 10 units.

b) Units processed by block A are routed equally to blocks B, C and D. During the first eight hours of the second cycle of phase P1, 10 units per hour are sent by block A to blocks B, C and D. Each of these blocks gets 3.33 units per hour. The capacity of each of these blocks is 5 units per hour. Thus, during the first eight hours of P1 in the second cycle, blocks B, C and D process 3.33 units per hour with an excess capacity of 1.67 units per hour. During the last two hours, block A processes 5 units per hour. Thus, blocks B, C and D process 1.67 units per hour with an excess capacity of 3.33 units per hour. At the end of the phase, blocks B, C and D each process a total of 30 units $(3.33 \times 8 + 1.67 \times 2 = 30)$with an excess capacity of 20 units $(1.67 \times 8 + 3.33 \times 2 = 20)$.

c) Blocks B, C and D route each of their units to block E. The capacity of E in this phase is 10 units per hour. Block E processes 10 units per hour with no excess capacity during the first eight hours of the phase. During the last two hours, block E processes 5 units per hour with an excess capacity of 5 units per hour. By the end of the phase, block E processes a total of 90 units with an excess capacity of 10 units.

A summary result table for the second cycle of phase P1 is shown next.



Block Throughput Summary

Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	90	10	0	40	0
B	30	20	0	0	0
C	30	20	0	0	0
D	30	20	0	0	0
E	90	10	0	0	0

5. Phase P2 in the second cycle from 40 to 50 hours

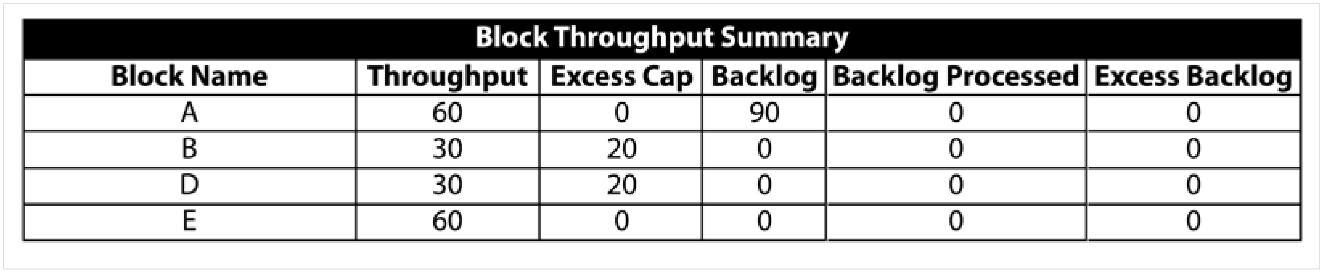
    a) As in the first cycle, phase P2 sends 15 units in its first hour to block A. The capacity of block A in this phase is 6 units per hour. Thus, block A processes only 6 units while there is a backlog of 9 units per hour.

    b) The 6 units processed by block A are routed equally to blocks B and D, as C is absent in this phase. Blocks B and D receive 3 units each. The capacity of each of these blocks is 5 units per hour. Thus, in the second cycle, blocks B and D process 3 units each with an excess capacity of 2 units each in the first hour of the second phase.

    c) Blocks B and D route a total of 6 units to block E. The capacity of E in this phase is 6 units per hour. Thus, block E processes 6 units in the first hour of the phase with no excess capacity and no backlog.

    d) The above steps are repeated for the ten hours of phase P2. At 50 hours, phase P2 ends in the second cycle.

A summary result table for the second cycle of phase P2 is shown next.



Block Throughput Summary

Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	60	0	90	0	0
B	30	20	0	0	0
D	30	20	0	0	0
E	60	0	0	0	0

6. Phase P3 in the second cycle from 50 to 60 hours

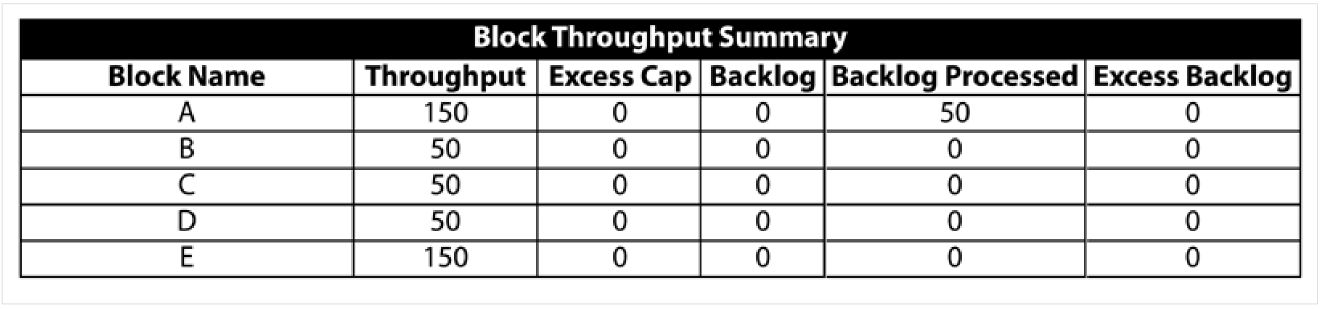
    a) Phase P3 sends 10 units in its first hour to block A in the second cycle. The capacity of block A in this phase is 15 units per hour. Thus, block A is able to process all 10 units sent by P3 and also process 5 units of backlog from the second cycle of phase P2. Thus, during the first hour of phase P3, block A processes 15 units that includes a backlog of 5 units. Since block A is used to its full capacity, the excess capacity of A is zero.

    b) The 15 units processed by block A in the first hour are routed equally over three paths to blocks B, C and D, as block C is available again in this phase. Blocks B, C and D receive 5 units each. The capacity of each of these block is 5 units per hour. Thus, blocks B, C and D process 5 units each to full capacity with no excess capacity during the first hour of the third phase.

    c) Blocks B, C and D route each of their 5 units to block E. The capacity of E in this phase is 15 units per hour. Thus, block E processes 15 units in the first hour of phase P3 with no excess capacity and no backlog.

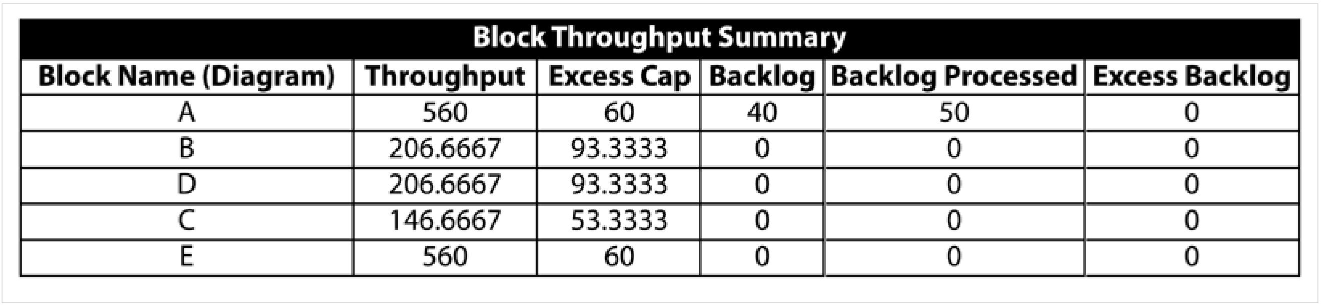
    d) The above steps are repeated for the ten hours of phase P3. At 60 hours, phase P3 ends. This ends the second cycle of the phase diagram and also marks the end of the simulation.

A summary result table for the second cycle of phase P3 is shown next.



Block Throughput Summary					
Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	150	0	0	50	0
B	50	0	0	0	0
C	50	0	0	0	0
D	50	0	0	0	0
E	150	0	0	0	0

A summary result table for the overall system for the simulation from 0 to 60 hours is shown in the following table.



Block Throughput Summary					
Block Name (Diagram)	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	560	60	40	50	0
B	206.6667	93.3333	0	0	0
D	206.6667	93.3333	0	0	0
C	146.6667	53.3333	0	0	0
E	560	60	0	0	0

Variable (Time-Varying) Throughput

Time-varying throughput can be specified at the phase level through the Variable Throughput property of an operational phase. Variable throughput permits modeling of scenarios where the throughput changes over time. For variable throughput, three general models are available in BlockSim. Each of these models has two parameters *a* and *b* which are specified by the user. These models are discussed below:

1. Linear model:

$$y = ax + b$$

This model describes the change in throughput $y$as a linear function of time x. Throughput processed between any two points of time $x_1$and $x_2$is obtained by integration of the linear function as:

$$\begin{aligned}\text{Linearly varying throughput} &=_{x_1}^{x_2} y dx \\ &=_{x_1}^{x_2} (ax+b)dx \\ &=\frac{a}{2}(x_2^2 - x_1^2) + b(x_2 - x_1)\end{aligned}$$

2. Exponential model:

$$y = be^{ax}$$

This model describes the change in throughput $y$as an exponential function of time x. Throughput processed in a period of time between any two points $x_1$and $x_2$is obtained as:

$$\begin{aligned}\text{Exponentially varying throughput} &=_{x_1}^{x_2} y dx \\ &=_{x_1}^{x_2} be^{ax} dx \\ &= \frac{b}{a}(e^{ax2} - e^{ax1})\end{aligned}$$

3. Power model:

$$y = bx^a$$

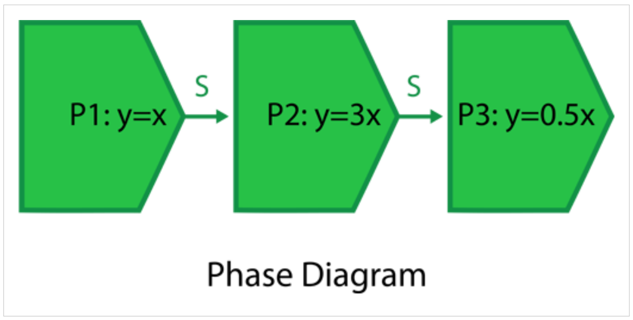
This model describes the change in throughput $y$as a power function of time x. Throughput processed between two points of time $x_1$and $x_2$is obtained as:

$$\begin{aligned}\text{Power varying throughput} &= {}_{x1}^{x2} y dx \\ &= {}_{x1}^{x2} bx^a dx \\ &= \frac{b}{a+1}(x_2^{a+1} - x_1^{a+1})\end{aligned}$$

All of the above models also have a user defined maximum throughput capacity value. Once this maximum throughput capacity value is reached, the throughput per unit time becomes constant and equal in value to the maximum throughput capacity specified by the user. In this situation, the variable throughput model would then act as a constant throughput model. The above models may at first glance seem limited, when in fact they do provide ample modeling flexibility. This flexibility is achieved by using these functions as building blocks for more complex functions. As an example, a step model can be easily created by using multiple phases, each with a constant throughput. A ramp model would use phases with linearly increasing functions in conjunction with constant phases, and so forth.

Variable Throughput Example

To examine variable throughput in phase diagrams, consider the phase diagram used above in the constant throughput example. All phase properties and linked RBDs remain unchanged for this illustration except that all the three phases now use variable throughput models. For the first phase, P1, the variable throughput model used is $y = x$(which is the linear model $y = ax + b$with parameters $a = 1$and $b = 0$). For the second phase, P2, the model used is $y = 3x$(which is the linear model $y = ax + b$with parameters $a = 3$and $b = 0$). The third phase uses $y = 0.5x$as the variable throughput model (which is the linear model $y = ax + b$with parameters $a = 0.5$and $b = 0$). Phases P1 and P2 have a value of 500 units per hour set for the maximum throughput capacity. This relatively larger value of throughput is set for the two phases so that the phases do not reach their maximum throughput value during the simulation. Phase P3 has a value of 4 units per hour set as the maximum throughput capacity. As in the previous example, all phases have a duration of ten hours. For the sake of simplicity it is assumed that the blocks can never fail and that items are routed equally to each path. The ignore backlog option is left unchecked for all blocks in the phase diagram.



If a throughput simulation with an end time of 60 hours is run on the phase diagram, the following scenario can be observed:

1. Phase P1 from 0 to 10 hours

   a) The throughput from phase P1 follows a time-varying model given by $y = ax + b$with parameters $a = 1$and $b = 0$. Thus, during the first hour the throughput sent by phase P1 to block A is:

$$\begin{aligned} Phasethroughputfrom0to1hr &= \frac{a}{2}(x_2^2 - x_1^2) + b(x_2 - x_1) \\ &= \frac{1}{2}(1^2 - 0^2) + 0(1 - 0) \\ &= 0.5\ units \end{aligned}$$

And the throughput sent by phase P1 to block A in the second hour is:

$$\begin{aligned} Phasethroughputfrom1to2hrs &= \frac{a}{2}(x_2^2 - x_1^2) + b(x_2 - x_1) \\ &= \frac{1}{2}(2^2 - 1^2) + 0(2 - 1) \\ &= 1.5 \ units \end{aligned}$$

It can be seen that the throughput from phase P1 sent to block A increases with time. But the capacity of block A remains constant at 10 units per hour during the entire phase P1. Thus, it becomes important to know the point of time when the number of units sent by P1 exceed the capacity of block A and the block starts accumulating a backlog. This can be obtained by solving the throughput model for phase P1 $(y = x)$using the capacity of block A by substituting $y$as the block capacity as shown next:

$$\begin{aligned} y &= x \\ or \ 10 &= x \end{aligned}$$

Thus, at time $x = 10$hours the number of units supplied per hour by phase P1 will equal the capacity of block A. At this point of time block A will function to its full capacity with no excess capacity and no backlog. Beyond this point the units sent by phase P1 will exceed the capacity of block A and the block will start accumulating a backlog. Since the duration of phase P1 is 10 hours, backlog at block A will not be a concern in this phase because up to this time the capacity of the block exceeds the units sent by phase P1. Thus, the total units sent by phase P1 to block A during the phase duration of 10 hours is:

$$\begin{aligned} &= \frac{a}{2}(x_2^2 - x_1^2) + b(x_2 - x_1) \\ &= \frac{1}{2}(10^2 - 0^2) + 0(10 - 0) \\ &= 50 \ units \end{aligned}$$

The total number of units that can be processed by block A (or the capacity of A) during phase P1 is:

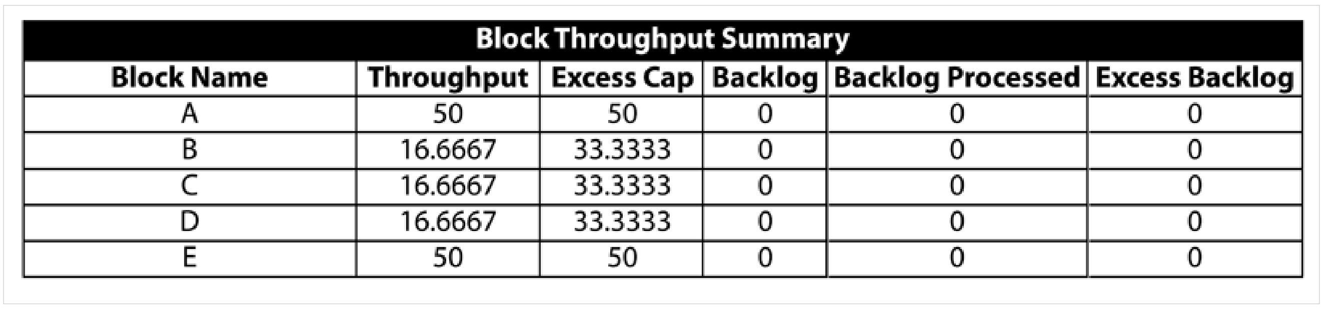
$$\begin{aligned} &= 10 \times 10 \\ &= 100 \ units \end{aligned}$$

Thus, during the first phase, block A processes all 50 units sent by phase P1 with an excess capacity of 50 units (100-50=50 units).

b) The 50 units processed by block A from time 0 to 10 hours are routed equally over three paths to blocks B, C and D. Each of these blocks gets a total of 16.67 units from time 0 to 10 hours. The capacity of each of these blocks is 5 units per hour. Thus, the total capacity of each of the blocks from time 0 to 10 hours is 50 units. As a result, each of the three blocks processes 16.67 units in phase P1 with an excess capacity of 33.33 units.

c) Blocks B, C and D route each of their 16.67 units to block E. The capacity of E in this phase is 10 units per hour or 100 units for the ten hours. Thus, block E processes a total of 50 units from blocks B, C and D with an excess capacity of 50 units during the first phase P1. At 10 hours, phase P1 ends and phase P2 begins.

A summary result table for phase P1 is shown next.



Block Throughput Summary					
Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	50	50	0	0	0
B	16.6667	33.3333	0	0	0
C	16.6667	33.3333	0	0	0
D	16.6667	33.3333	0	0	0
E	50	50	0	0	0

2. Phase P2 from 10 to 20 hours

a) The throughput of phase P2 follows the model $y = ax + b$with $a = 3$and $b = 0$. The throughput from P2 goes to block A. The capacity of block A in this phase is 6 units per hour. The point of time when the number of units sent by P2 equals the capacity of block A can be obtained by substituting the capacity of block A into the model equation $(y = 3x)$as shown next:

$$\begin{aligned} y &= 3x \\ or\ 6 &= 3x \\ or\ x &= 2 \end{aligned}$$

Thus, at $x = 2$hours of phase P2 (or at the total simulation time of 12 hours), the number of units sent per hour by phase P2 equals the capacity of block A. Thus, during the first two hours of phase P2, block A will have excess capacity. The total number of units sent by phase P2 during the first two hours of the phase is:

$$\begin{aligned} &= \frac{a}{2}(x_2^2 - x_1^2) + b(x_2 - x_1) \\ &= \frac{3}{2}(2^2 - 0^2) + 0(2 - 0) \\ &= 6\ units \end{aligned}$$

The total capacity of block A during the first two hours of phase P2 is:

$$\begin{aligned} &= 6 \times 2 \\ &= 12\ units \end{aligned}$$

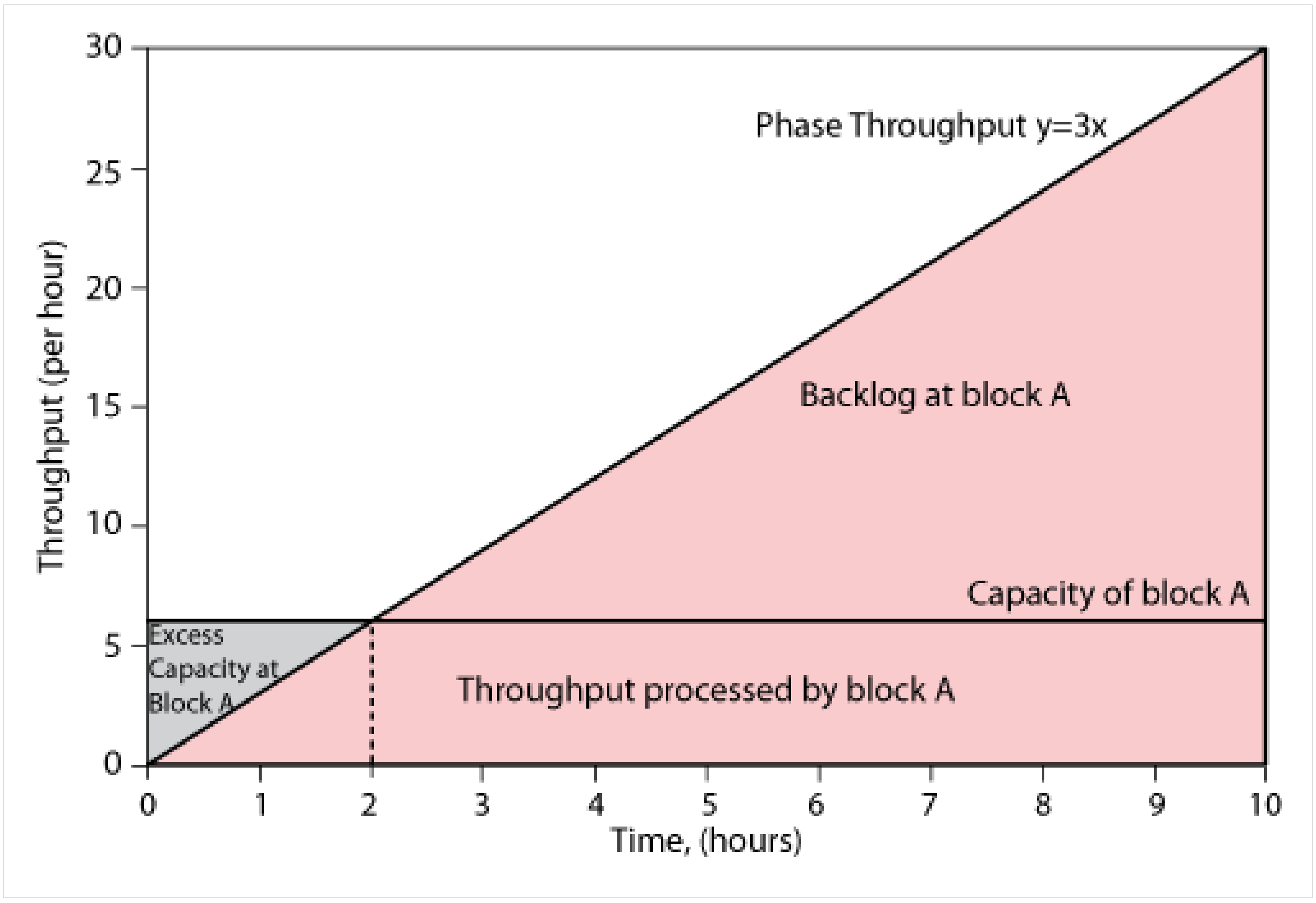
Thus block A processes a total of 6 units in the first two hours of phase P2 with an excess capacity of 6 units (12-6=6 units). During the last eight hours of the phase (from the second hour to the tenth hour) block A will have a backlog because the number of units sent by phase P2 will exceed block A's capacity. The total number of units sent by phase P2 during the last eight hours of the phase is:

$$\begin{aligned} &= \frac{a}{2}(x_2^2 - x_1^2) + b(x_2 - x_1) \\ &= \frac{3}{2}(10^2 - 2^2) + 0(10 - 2) \\ &= 144\ units \end{aligned}$$

The total capacity of block A during these eight hours of phase P2 is:

$$\begin{aligned} &= 6 \times 8 \\ &= 48\ units \end{aligned}$$

Thus, during the last eight hours of phase P2, block A will process a total of 48 units with a backlog of 96 units (144-48=96 units). A graphical representation of the above scenario for block A during phase P2 is shown in the figure below.



b) The units processed by block A are routed equally to blocks B and D, as block C is absent in this phase. During the first two hours, block A sends a total of 6 units that are divided equally among blocks B and D, with each block getting 3 units. The capacity of these blocks in this phase is 5 units per hour. Thus the total capacity of each of these blocks during the first two hours of phase P2 is:

$$=5 \times 2$$
$$=10 \; units$$

As a result, blocks B and D each process 3 units in the first two hours of phase P2 with an excess capacity of 7 units (10-3=7 units). During the last eight hours of phase P2, block A sends a total of 48 units to blocks B and D, with each block receiving 24 units. The capacity of each of these blocks during these eight hours is:

$$=5 \times 8$$
$$=40 \; units$$

Thus, B and D process 24 units in the last eight hours of phase P2, with an excess capacity of 16 units (40-24=16 units).

c) The units processed by blocks B and D are sent to block E. The capacity of E in this phase is 6 units per hour. During the first two hours of phase P2, a total of 6 units are sent to block E. The total capacity of block E during this period of time is:

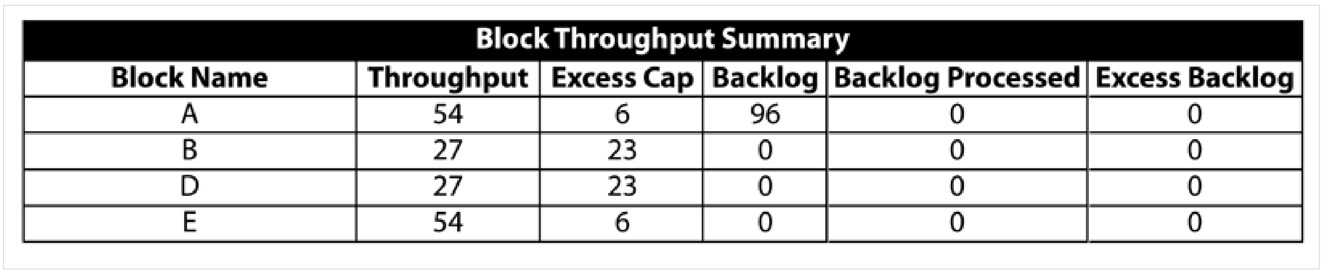
$$=6 \times 2$$
$$=12 \; units$$

Consequently block E processes 6 units in the first two hours of phase P2 with an excess capacity of 6 units (12-6=6 units). During the last eight hours of the phase, blocks B and D send a total of 48 units to block E. The capacity of block E during this time is:

$$=6 \times 8$$
$$=48 \ units$$

Thus, block E processes 48 units in the last eight hours of P2 with no excess capacity and no backlog. At 20 hrs phase P2 ends and phase P3 begins.

A summary result table for phase P2 is shown next.



Block Throughput Summary					
Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	54	6	96	0	0
B	27	23	0	0	0
D	27	23	0	0	0
E	54	6	0	0	0

3. Phase P3 from 20 to 30 hours

a) The throughput of phase P3 follows the model $y = ax + b$with $a = 0.5$and $b = 0$. A maximum throughput capacity of 4 units per hour is also set for this phase. The throughput from this phase goes to block A. The capacity of block A in this phase is 15 units per hour. Thus, it can be seen that the throughput sent by phase P3 will never exceed the capacity of block A in this phase. It can also be seen that at some point of time the throughput of phase P3 will reach the maximum throughput capacity of 4 units per hour and, thereafter, a constant throughput of 4 units per hour will be sent by phase P3. This point of time can be calculated by solving the linear throughput model for the phase $y = 0.5x$by substituting $y$as the maximum throughput capacity as shown next:

$$y = 0.5x$$
$$or \ 4 = 0.5x$$
$$or \ x = 8$$

Thus, at time $x = 8$hours of phase P3 (or at the time of 28 hours) the number of units sent per hour by phase P3 will reach the constant value of 4 units per hour. In the first eight hours of the phase the throughput will remain variable. The total number of units sent by P3 to block A during this time is given by:

$$=\frac{a}{2}(x_2^2 - x_1^2) + b(x_2 - x_1)$$
$$=\frac{0.5}{2}(8^2 - 0^2) + 0(8 - 0)$$
$$=16 \ units$$

The total capacity of block A during the first eight hours of phase P3 is:

$$=15 \times 8$$
$$=120 \ units$$

Consequently, it can be seen that block A has an excess capacity of 104 units (120-16=104 units). Thus, block A will be able to process all of the 96 units of backlog from phase P2 during these eight hours. As a result, block A will process a total of 112 units (16 units from P3 and 96 units from the backlog of phase P2) with an excess capacity of 8 units (120-112=8 units) during the first eight hours of phase P3. In the last two hours of this phase, a constant throughput of 4 units per hour is sent to block A, thus the total number of units sent to the block during the last two hours are:

$$=4 \times 2$$
$$=8 \ units$$

The capacity of block A during these two hours is:

$$=15 \times 2$$

$$=30\ units$$

Thus, block A will process 8 units in the last two hours of phase P3 with an excess capacity of 22 units (30-8=22 units).

b) The units processed by block A are routed equally over three paths to blocks B, C and D, as block C is available again in this phase. During the first eight hours of phase P3 a total of 112 units are sent to blocks B, C and D, with each block receiving 37.33 units. The capacity of each of these block is 5 units per hour, thus the total capacity of each of the blocks during the first eight hours is:

$$=5 \times 8$$

$$=40\ units$$

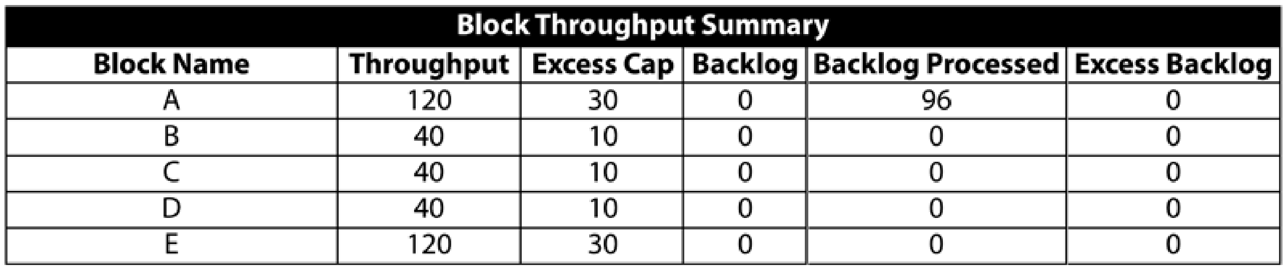
As a result, blocks B, C and D each process 37.33 units in the first eight hour of phase P3 with an excess capacity of 2.67 units (40-37.33=2.67 units). In the last two hours of the phase, block A sends a total of 8 units, with each block getting 2.67 units. The capacity of blocks B, C or D during these two hours is:

$$=5 \times 2$$

$$=10\ units$$

Thus, blocks B, C and D each process 2.67 units in the last two hours of phase P3 with an excess capacity of 7.33 units (10-2.67=7.33 units). At the end of these two hours phase P3 gets completed. This also marks the end of the simulation.

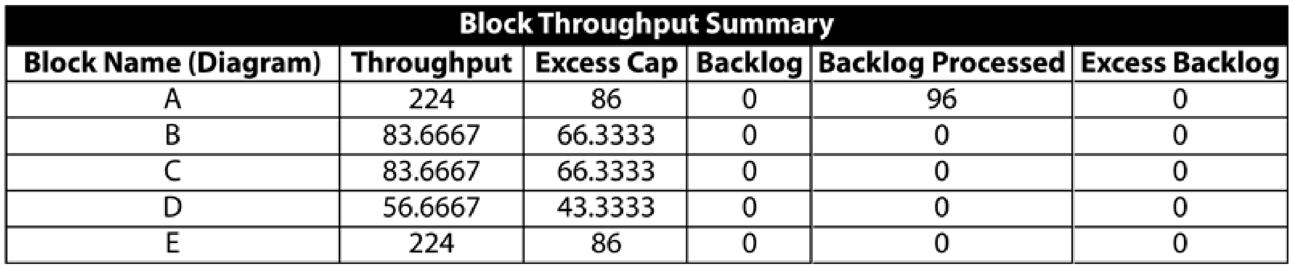
A summary result table for phase P3 is shown next.



Block Throughput Summary

Block Name	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	120	30	0	96	0
B	40	10	0	0	0
C	40	10	0	0	0
D	40	10	0	0	0
E	120	30	0	0	0

A summary result table for the overall system for the simulation from 0 to 30 hrs is shown in the following table.



Block Throughput Summary

Block Name (Diagram)	Throughput	Excess Cap	Backlog	Backlog Processed	Excess Backlog
A	224	86	0	96	0
B	83.6667	66.3333	0	0	0
C	83.6667	66.3333	0	0	0
D	56.6667	43.3333	0	0	0
E	224	86	0	0	0

Appendices

Appendix A: Generating Random Numbers from a Distribution

Simulation involves generating random numbers that belong to a specific distribution. We will illustrate this methodology using the Weibull distribution.

Generating Random Times from a Weibull Distribution

The *cdf* of the 2-parameter Weibull distribution is given by:

$$F(T) = 1 - e^{-\left(\frac{T}{\eta}\right)^{\beta}}$$

The Weibull reliability function is given by:

$$\begin{aligned} R(T) &= 1 - F(t) \\ &= e^{-\left(\frac{T}{\eta}\right)^{\beta}} \end{aligned}$$

To generate a random time from a Weibull distribution, with a given η and β a uniform random number from 0 to 1, $U_R[0,1]$, is first obtained. The random time from a weibull distribution is then obtained from:

$$T_R = \eta \cdot \{-\ln\left[U_R[0,1]\right]\}^{\frac{1}{\beta}}$$

Conditional

The Weibull conditional reliability function is given by:

$$R(T,t) = \frac{R(T+t)}{R(T)} = \frac{e^{-\left(\frac{T+t}{\eta}\right)^{\beta}}}{e^{-\left(\frac{T}{\eta}\right)^{\beta}}}$$

BlockSim's Random Number Generator (RNG)

Internally, ReliaSoft's BlockSim uses an algorithm based on L'Ecuyer's [RefX] random number generator with a post Bays-Durham shuffle. The RNG's period is approximately 10^18. The RNG passes all currently known statistical tests, within the limits the machine's precision, and for a number of calls (simulation runs) less than the period. If no seed is provided the algorithm uses the machines clock to initialize the RNG.

References

1. L'Ecuyer, P., 1988, Communications of the ACM, vol. 31, pp.724-774
2. L'Ecuyer, P., 2001, Proceedings of the 2001 Winter Simulation Conference, pp.95-105
3. William H., Teukolsky, Saul A., Vetterling, William T., Flannery, Brian R., Numerical Recipes in C: The Art of Scientific Computing, Second Edition, Cambridge University Press, 1988.
4. Peters, Edgar E., Fractal Market Analysis: Applying Chaos Theory to Investment & Economics, John Wiley & Sons, 1994.
5. Knuth, Donald E., The Art of Computer Programming: Volume 2 - Seminumerical Algorithms, Third Edition, Addison-Wesley, 1998.

Appendix B: References

1. Aitchison, J. and J.A.C. Brown, The Lognormal Distribution, Cambridge University Press, New York, 1957.
2. Barlow, R. and L. Hunter, Optimum Preventive Maintenance Policies, Operations Research, Vol. 8, pp. 90-100, 1960.
3. Cramer, H., Mathematical Methods of Statistics, Princeton University Press, Princeton, NJ, 1946.
4. Davis, D.J., An Analysis of Some Failure Data, J. Am. Stat. Assoc., Vol. 47, 1952.
5. Dietrich, D., SIE 530 Engineering Statistics Lecture Notes, The University of Arizona, Tucson, Arizona.
6. Dudewicz, E.J. and S.N. Mishra, Modern Mathematical Statistics, John Wiley & Sons, Inc., New York, 1988.
7. Elsayed, E., Reliability Engineering, Addison Wesley, Reading, MA, 1996.
8. Hahn, G.J. and S.S. Shapiro, Statistical Models in Engineering, John Wiley & Sons, Inc., New York, 1967.
9. Kapur, K.C. and L.R. Lamberson, Reliability in Engineering Design, John Wiley & Sons, Inc., New York, 1977.
10. Kececioglu, D., Reliability Engineering Handbook, Prentice Hall, Inc., New Jersey, 1991.
11. Kececioglu, D., Maintainability, Availability, & Operational Readiness Engineering, Volume 1, Prentice Hall PTR, New Jersey, 1995.
12. Kijima, M. and Sumita, N., A useful generalization of renewal theory: counting process governed by nonnegative Markovian increments, Journal of Applied Probability, 23, 71--88, 1986.
13. Kijima, M., Some results for repairable systems with general repair, Journal of Applied Probability, 20, 851--859, 1989.
14. Knuth, D.E., The Art of Computer Programming: Volume 2 - Seminumerical Algorithms, Third Edition, Addison-Wesley, 1998.
15. L'Ecuyer, P., Communications of the ACM, Vol. 31, pp.724-774, 1988.
16. L'Ecuyer, P., Proceedings of the 2001 Winter Simulation Conference, pp.95-105, 2001.
17. Leemis, L.M., Reliability - Probabilistic Models and Statistical Methods, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1995.
18. Lloyd, D.K. and M. Lipow, Reliability: Management, Methods, and Mathematics, Prentice Hall, Englewood Cliffs, New Jersey, 1962.
19. Mann, N.R., R.E. Schafer and N.D. Singpurwalla, Methods for Statistical Analysis of Reliability and Life Data, John Wiley & Sons, Inc., New York, 1974.
20. Meeker, W.Q. and L.A. Escobar, Statistical Methods for Reliability Data, John Wiley & Sons, Inc., New York, 1998.
21. Mettas, A., Reliability Allocation and Optimization for Complex Systems, Proceedings of the Annual Reliability & Maintainability Symposium, 2000.
22. Nelson, W., Applied Life Data Analysis, John Wiley & Sons, Inc., New York, 1982.
23. Peters, E.E., Fractal Market Analysis: Applying Chaos Theory to Investment & Economics, John Wiley & Sons, 1994.

24. Press, W.H., S.A. Teukolsky, W.T. Vetterling and B.R. Flannery, Numerical Recipes in C: The Art of Scientific Computing, Second Edition, Cambridge University Press, 1988.
25. ReliaSoft Corporation, Life Data Analysis Reference, ReliaSoft Publishing, Tucson, Arizona, 2005.
26. Tillman, F.A., C.L. Hwang and W. Kuo, Optimization of Systems Reliability, Marcel Dekker, Inc., 1980.
27. Weibull, W., A Statistical Representation of Fatigue Failure in Solids, Transactions on the Royal Institute of Technology, No. 27, Stockholm, 1949.
28. Weibull, W., A Statistical Distribution Function of Wide Applicability, Journal of Applied Mechanics, Vol. 18, pp. 293-297, 1951.