Knowing When to Say When

Abstract:
Developing software is easy. It is just a matter of ones and zeros. The hard part of developing software is putting those ones and zeros in the proper order to perform useful work. Testing is one of the primary methods of demonstrating that the ones and zeros are in the proper order. One the most perplexing software testing problems is knowing when sufficient testing has been performed to justify the risk of shipping a software product to the field. This paper provides just such a methodology for resolving this problem. The methodology employs the proven Weibull distribution, binomial distributions, and Markov analysis . With a deterministic nature, software failure distributions do not have the sensitivity to random variables. The paper provides an adroit approach for handling the deterministic nature of software. Coupling this approach with the proposed methodology a tool is developed. The paper concludes with a real world example employing the tool. The resulting tool can be applied to software applications of varying size and complexity. The paper closes with a discussion concerning issues which might "vacate" conclusions derived from this tool. Finally the paper contains an addendum. This addendum provides a procedural template for anyone interested in employing the approach. The addendum also provides the more adventurous individuals with the rigorous mathematical treatment and definitions.

Introduction:
No one ships problems to the customer. At least they don't for long before that marketing strategy collapses. Given that a software company has learned that shipping problems to the customer is a self-defeating strategy, that company employs some type of mechanism for determining the "quality" or "readiness" of a new product for shipment to the customer. The approach a company takes in measuring software quality depends upon its perspective[1]. A company's perspective can be the producer's view and the consumer's view.

Typically the producer's view is taken when measuring quality. This perspective results in quality measurement mechanisms which may range from Aunt Clara "banging away" until the pizza is gone to defect trend report threshold measurement to formal system level acceptance Go/NoGo testing. These various approaches concentrate on measuring the level of maturity of the new software at a specific point in time. These approaches fail to ascertain the expected performance level of the software in the customers' hands. An example of this is shown in Figure 1: Accumulated Defects by Week. This producer's view tends to demonstrate the level of activity as measured by accumulating defects. Using this view, it is difficult to derive a "feel" of whether the software is maturing.

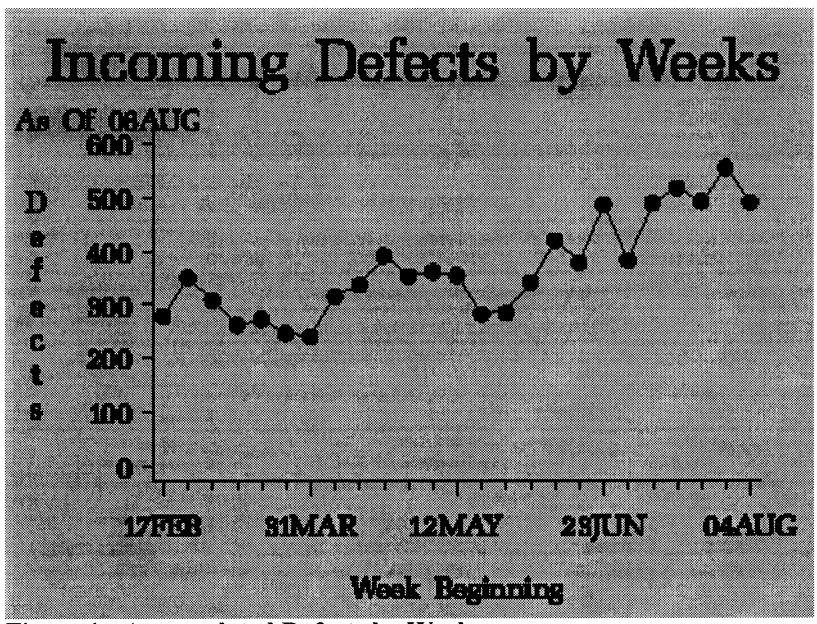


Figure 1: Accumulated Defects by Week

This paper examines an approach for establishing software maturity. The remainder of the paper will assume a view of software maturity from the customers perspective. This customer view is software reliability. The basic concepts of the approach will be introduced at the "10,000 foot level". The paper's addendum will provide the granularity detail to employ the approach at an engineering level. The basic concepts will be used to under a real world scenario to illustrate the approach. Finally, hybrids of this approach and vacating issues will be explored.

Software Reliability:
The producer's view fails to address problems with which the customer will have to "deal". Another way to say this that they fail to provide an indication of the likelihood of remaining problems that the customer will encounter. A better "customer view" of software quality can be obtained by measuring the reliability of the software. This reliability is the probability that the software will exhibit failure free operation for a specified duration for a defined operational profile.

Software reliability is a much more profound measure of quality. It is consumer oriented. It is dynamic in reflecting how the software is employed. It tracks the operation of the software and the resulting faults which are exhibited. It lends itself to tracking trends and forecasting performance.[2]

Software reliability tends to exhibit four characteristics. First, hardware and software components can be combined to determine system reliability. Second, the predominate source leading to software failure is caused by "design faults". Third, software does not degrade in any physical sense like hardware components. Fourth, software reliability changes continuously during the development/test period.[3]

Moving forward with measuring software reliability, this paper develops a Markovian approach for establishing software quality. The Markovian approach yields a forecast of the software performance level in the hands of the customer. The Markovian approach is not perfect, but it is rooted in solid software, testing, and mathematical methodology. The Markovian approach will introduce the concepts of randomness, the Weibull distribution, the binomial distribution, and the Markov analysis. The Markovian approach has a long history of successful application.[4]

Randomness:
The first concept employed by the Markovian approach is randomness. This approach draws a distinction between random and non-random failure events. For an event to be random, the chance that it will occur across a interval of interest is dependent on a random variable. The random variable represents a set of values each of which has an un-bias opportunity of selection. Flipping a silver dollar is generally thought of as an event which will randomly produce heads or tails.

Although hardware and software systems have similarities, one of the unique differences between the two is the deterministic nature of software failure events. This nature implies that for an explicit set of input conditions there is a unique set of expected output conditions for a software system.

In contrast hardware systems failures generally occur in four distinct groups. This can be demonstrated with the traditional "bath tub curve" as shown in Figure 2. Traditional Bath Tub Curve. This curve is plotted as time versus failure rate. The leading edge (infant mortality) of the curve shows a decreasing failure rate. The trailing edge (wearout) of the curve shows an increasing failure rate. The flat bottom portion of the curve displays a uniform or constant failure rate. Any abrupt spikes within the "bath tub curve" represent weak sub-populations. Failures occurring in leading edges, trailing edges, and spikes represent non-random failure events. Those occurring in the flat region are random failure events. In the physical world, no failure is truly random. That means that each hardware failure has a definite failure mechanism. What makes a failure event random is that it occurs at a randomly "drawn" duration.

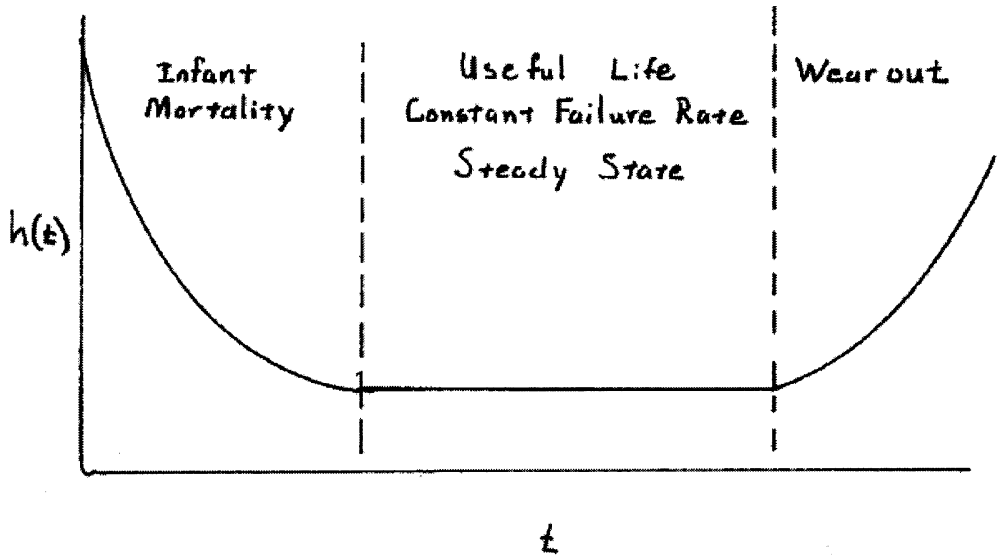


Figure 2: Traditional Bath Tub Curve

The question of providing a random "draw" duration for usage is solved using a strict definition. The Markovian approach uses the accumulation of individual test cases as opposed to time, clock cycles, etc as the definition of the unit of duration for the software. The test cases for this approach compose a test library. Each test cases must be completely independent of all other test cases. Test cases are randomly drawn from the test library until all test cases have been executed.

For example, assume that a test library is composed of twenty test cases. Of those twenty, two "A" and "B" are known to be extremely critical and used for "sanity checking". All twenty can be placed into the test library pool for random order drawing. Let us assume that the first of these two test cases, "A" was pulled in the fifteenth position and the second one, "B" in the fourth position. For "sanity checking" it is entirely appropriate to perform the two critical test cases first. However, when it comes for making any decision concerning software reliability, the results from these two test cases are evaluated in the order of the random draw.

Weibull Distribution:
The second concept used by the Markovian approach is that of the Weibull distribution. The Weibull distribution is a special way of looking at the information presented in the traditional "bath tub curve". The Weibull distribution is a more powerful version of the exponential distribution as shown in Equation 1: Weibull Failure Distribution of the Addendum. Using calculus on the Weibull distribution produces a graph of duration units along the x-axis and the cumulative probability of failure along the y-axis. See the addendum for a more rigorous presentation of this aspect. This style of graph is called cumulative distribution function.

Three aspects of the Weibull graph make it very useful. First, at the point were the scale parameter or characteristic life equals the duration yields a 63.2% cumulative probability of failure. The characteristic life is another term for mean time to failure. This provides a very useful line of demarcation. An example of this demarcation is shown in Figure 3. Weibull Plot Showing 63.2% Demarcation. In Figure 3, Group A is shown to have reached the characteristic life value about sixty days ahead of the composite All grouping. This strongly implies a Group A is significantly more mature than the rest of the composite All.

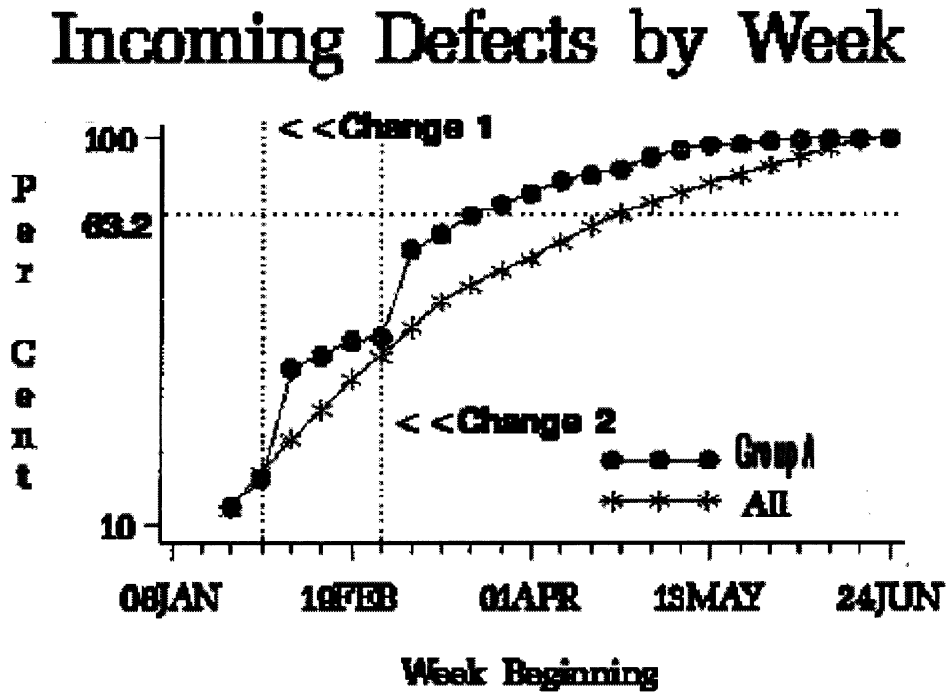


Figure 3: Weibull Plot Showing 63.2% Demarcation, Displaying 'S' Shaping, Group A Favorable to All

The second aspect that is useful is the distribution of various sub-populations of failures tend to group into easily recognizable clusters. Each of these clusters tends to form a clear "S" shape on the graph. This "S" shape has its own distinguishing slope. Combining these two aspects, it is possible to determine the mean time to failure for an entire populations of failures or for any of the various sub-populations. This feature of the Weibull graph readily supports failure analysis to the failure mechanism level.

The final aspect is that the plotting of the data is relatively straight forward. The data to be analyzed is plotted according to a media rank technique. The earlier test results tend to show the greatest "jumps" and "sharpest knees". The latter points tend to "smooth out" allowing refined forecasting and analysis. As test data is accumulated, statistically valid confidence intervals can be developed. Figure 4. Weibull Plot of Actual Software Test Data displays actual test data submitted to one of the many Weibull test analysis programs. [5] The early stair step appearance is due to the discreet nature of the random test case. A similar appearance is observed when hours, cycles, etc. is used. The characteristic life for this set of data suggest 7 test case between failure.

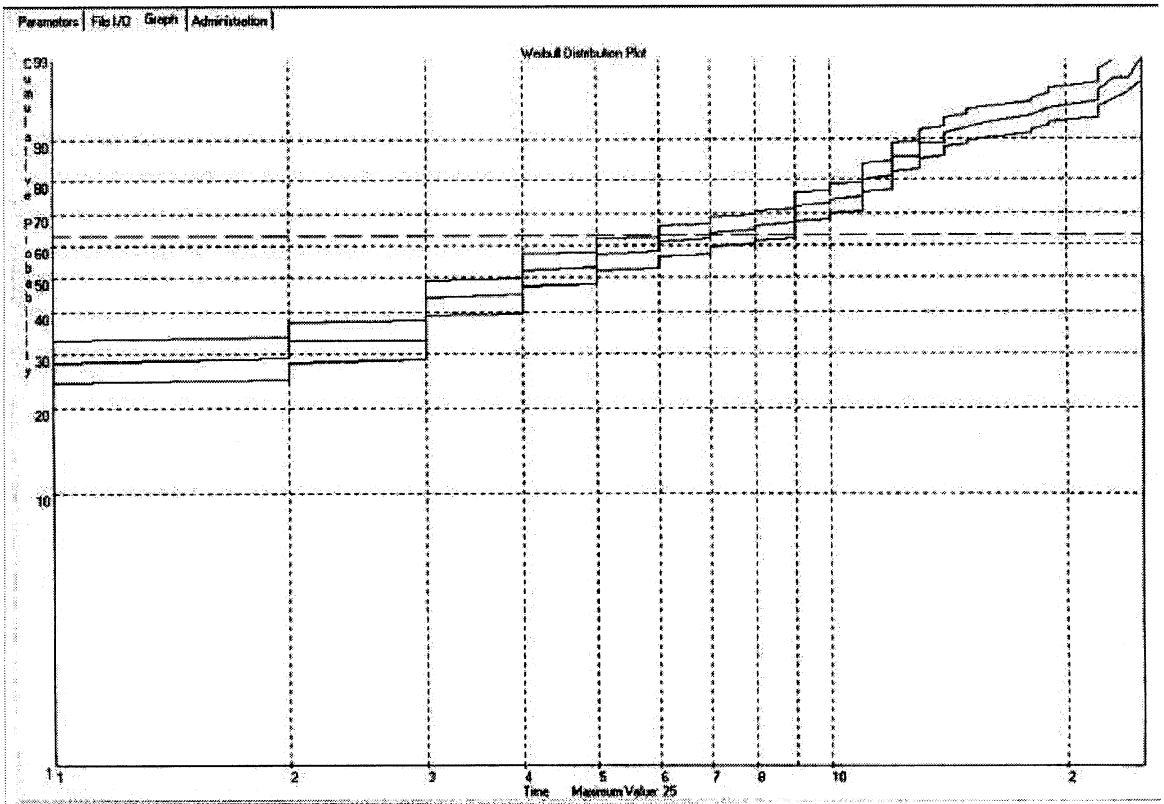


Figure 4: Weibull Plot of Actual Software Test Data with Confidence Bands Showing Characteristic Life of Seven

Binomial Distribution:

The binomial distribution is the third concept need by the Markovian approach. The binomial distribution is used to determine the outcomes of events. When software initially enters test, the number of inherent faults is not known. The lowest number is "0". The largest number is "N". However, if an estimate of the number of weak components can be made, an initial probability vector can be produced using this information and the Binomial distribution. This provides the probability of "0" faults, the probability of "1" fault, ..., the probability of "N" faults. The initial probability vector, Π_0, is just the ordered collection of these probabilities from 0 to N. See the definition of Π_0, in the addendum for a more rigorous treatment of the binomial distribution and the initial probability vector.

Markov Analysis:

Central to the Markovian approach is Markov analysis method. Markov analysis employs matrix multiplication. This multiplication involves three matrices: the input matrix, the transfer matrix, and the output matrix. A more rigorous treatment of each of these matrices and their elements is given in the addendum.

For the general purpose of this paper the three matrices can be thought of as shown in Figure 5: Analogy of a Markov Analysis. The initial matrix can be thought of as the toothpaste within an opened tube. We do not know how much toothpaste or defects we have in the opened tube that was just handed to us by development. The output matrix is represented by the target. We know that management wants no more than a than a teaspoon of toothpaste on the target. The transfer matrix is composed of the pneumatic workbench vice. Our tester activates the vice which squeezes the tube sending a quantity of toothpaste at the target.

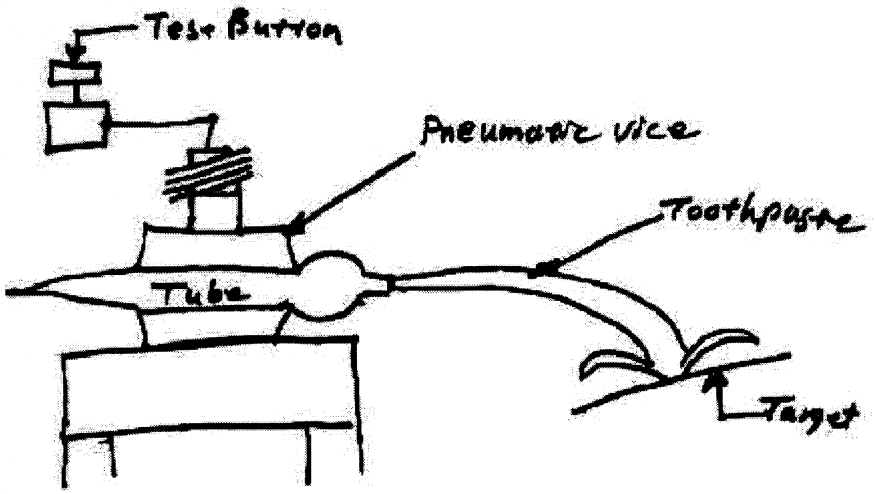


Figure 5: Analogy of a Markov Analysis, Toothpaste = Input Matrix, Target = Output Matrix, Pneumatic Vice = Transfer Matrix.

We test the toothpaste-vice-target analogy as shown in Figure 5.. If a test run results in too much toothpaste, we return the tube to development. They make an adjustment and send us a new tube of toothpaste. We re-test and adjust the toothpaste until the satisfactory amount is deposited on the target. When that point is reached we know how much toothpaste to put into the tube. We are then ready to ship.

Matrix multiplication of Markov analysis is not a trivial exercise. For simple systems with N=10, a spreadsheet analysis can be used to perform a Markov analysis. More complex systems {five to eighty components}can be analyzed using a tool such as AAP824 Software Test Planning/Analysis. [6] Even more complex systems can be analyzed using the graphs provided by Jensen/Petersen.[7]

Both AAP824 and Jensen/Petersen employ the same x-axis and y-axis. The y-axis serves as the target or output matrix. The diagonal lines serve as the vice or transfer matrix. The x-axis is roughly equivalent to the input matrix. It is actually a normalized unit-less value called "A" which will be discussed more completely later in the paper.

Figure 6: AAP824 2% Weak shows a simple software system of five components. In this instance a normalized value of A equal to 6 would yield a 99.7% chance that there were no failures remaining (or a .3% of one or more failures remaining).

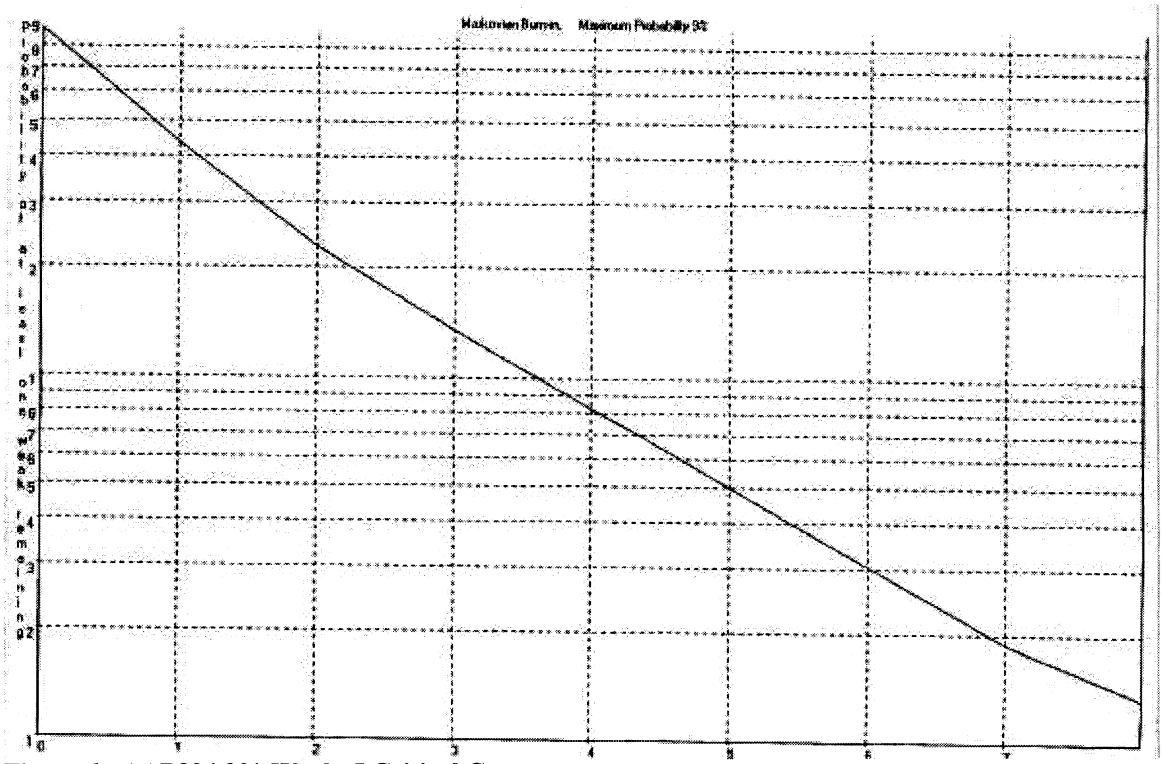


Figure 6: AAP824 2% Weak, 5 Critical Components.

Figure 7: Petersen/Jensen Markovian Graph Extended provides the capability to analyze more complex software systems. In this instance, a system with 10,000 components would need to demonstrate a A normalized value of at least 6.8 to yield a 99% likelihood of zero failures remaining (1% chance of one or more failures remaining).

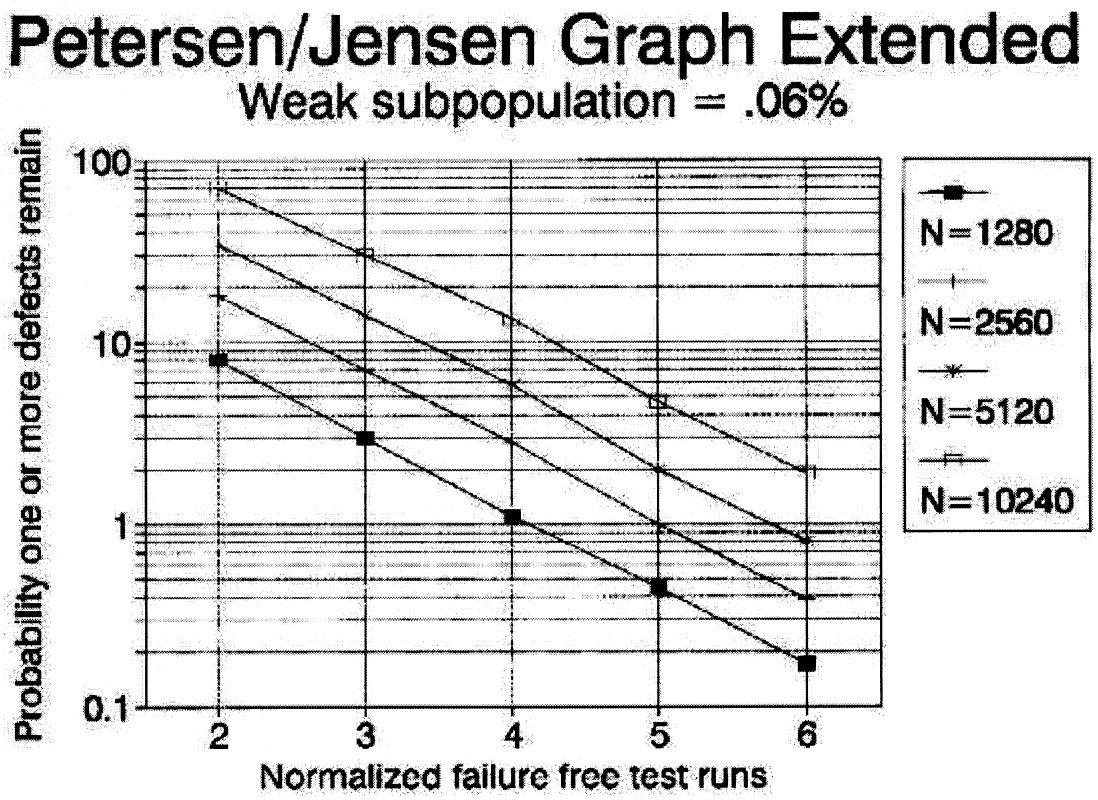


Figure 7: Petersen/Jensen Markovian Graph Extended to 10,240 Critical Components.[8]

Obtaining the Salient Failure Free Value:
The x-axis of the graph used for the Markovian approach is given by "A". "A" is the normalization of the "failure free period" and the characteristic life of the entire test case pool. The "A" can be found by locating the "decreed" performance level on the appropriate Markovian Graph such as Figure 5 or Figure 6, moving across the corresponding horizon line until intersecting the appropriate critical components line. Dropping vertically from that intersection to the x-axis locates the corresponding "A" value.

With the "A" normalized value identified and the characteristic life either estimated or calculated from test data, the key value "failure free period" can then be directly determined. As an example assume that testing has produced data supporting a characteristic life of 7 as shown in Figure 4. Further assume that the "decreed" performance level is 1% with 2560 critical components identified. These two last assumptions yield an "A" normalized value of 5. Solving for the failure free period ($T_M = A \bullet MTTF$) $T_M = 7 \bullet 5 = 35$.

The "failure free period" is the salient parameter of the Markovian approach. The "failure free period" or T_M represents the progress of the software testing against the randomly selected test cases. This tracking is shown in Figure 8: Failure Free Period. Using the example given above, assume that in Try 1, the failure was observed on the twenty-second test case. For Try 2, assume that the failure occurred during the nineteenth test case. Following each of these tries there was assumed to be a "repair" to the software. Finally, on Try 3 the software successfully negotiated a previously established failure free period of thirty-five. At that point it can be stated that the software has demonstrate sufficient maturity to yield the "decreed" field performance level of %1.

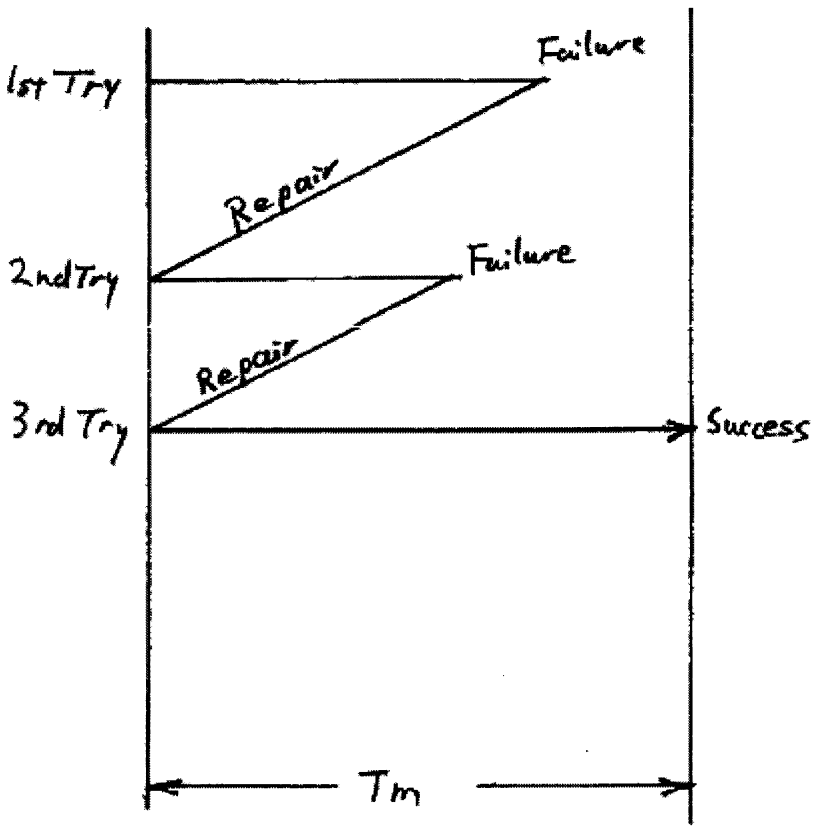


Figure 8: Failure Free Period

Software components:
The final key point to resolve in the application of the Markovian approach is the definition of "critical component". To grasp the concept of critical components it is essential to understand the composition of a system. A system should be viewed as a set of components as shown in Figure 9: Software System. This set of components is composed of a subset of components call critical components. The critical components derive their name from their perceived failure behavior propagating a failure to system behavior level. The "weak components" represent a subset of the critical components. A weak component actually surfaces a failure during testing which propagates to system level.

Unlike hardware, there is tremendous latitude in how an actual component can be defined. The Markovian approach has been successfully employed on occasions when components were defined as programming units as well as individual lines of code. The additional granularity gained in using the lines of code forced a corresponding increase in the complexity of Markovian approach. Although it has not yet been verified, it is very likely that individual machine instructions could also be defined as a component. This level of granularity would have a corresponding increase in analysis complexity.

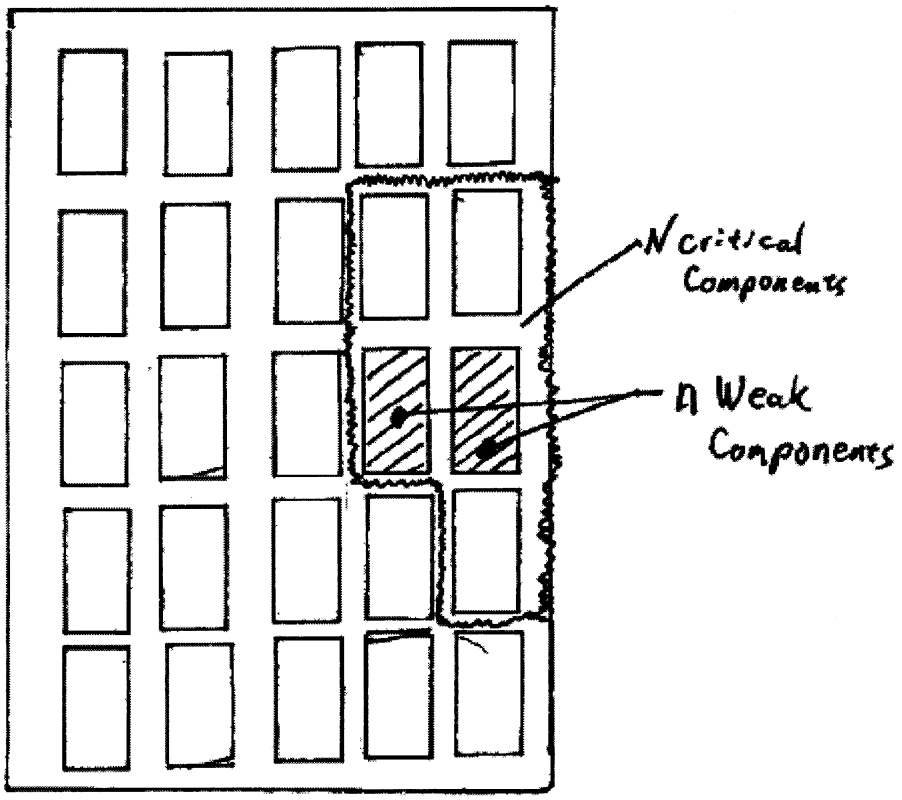


Figure 9: Software System, Critical Components, Weak Components

Alternate Approaches:
Although an analysis can be performed using AAP824 or the Jensen/Petersen extended Markovian plots, these tools begin to fall short as resources and granularity are stressed. One possible approach to this issue is the usage of a "complete natural response" methodology. This approach promises to readily handle analysis involving millions of critical components. Various software partners have been solicited to engage in this research effort.

Vacating Issues:
As with any process technique there are ways to "cheat". There are three obvious approaches that must be monitored. The first approach is "rounding liberally/conservatively". Those performing the analysis need to employ a set of "rounding" rules which lead to a fair, repeatable interpretation of the data. The second approach is bias test cases. The entire domain of test cases need not be employed during this effort. However, a fair representation of the operation profile, boundaries, exceptions, etc. test case conditions must be in the test case domain. The third approach is "non-relevant test cases". This approach seeks to generate favorable test numbers with little risk of test conditions actually forcing a failure. In this approach testing of new code is avoided by not flexing the critical components or the interfaces with them.

Summary:
The bottom line is that if a field performance level can be expressed, then a Markovian approach can be designed which will guarantee this level. The tools are available for modestly complex systems. The scalability of the Markovian approach to vary complex systems is yet to be thoroughly verified. The Markovian approach is not fool proof, but is does work. It has proven itself to be an acceptable, repeatable approach to determining when sufficient testing has been performed.

Further Reading:
The Petersen and Jensen's book titled Burn-in An Engineering Approach introduces the Markovian approach for hardware. Their book should be read in the order of chapter 9, chapter 6, and then chapter 3. The graphs and tables are extensible to large complex systems.

The Kapour and Lamberson's book title Reliability in Engineering Design provides one of the best discussions of Weibull distribution in chapter 11. This book also contains solid information on the Binomial distribution, hypothesis testing, and basic reliability concepts. The remainder of the book deals with hardware reliability issues.

The Musa, Iannino, and Okumto's book titled Software Reliability gives a classical presentation of software reliability. Chapter 1 provides a common framework for a discussion of software reliability. Chapters 10 and 11 provide theory for the binomial distribution, Weibull distribution, and Markovian models.

Special Thanks:
Thanks go to Professor Mladen Vouk of North Carolina State University. His course CSC691T: Software Testing and Reliability provided a solid foundation on which to begin a software testing career. He is also to be thanked for reviewing and providing comments concerning this paper. Finally, thanks go to the Research Triangle Park- Software Process Improvement Network for encouraging creation of this paper.

Addendum:
The addendum contains background information. This information provides a generic procedure template, a complete set of working definitions, equations, a derivation of the Weibull graph translation, an example for determining the Weibull slope between two points and a rigorous treatment of the Markov analysis as applied to this form of application.

Procedure Template:
This is a generic procedure which will guarantee a defined performance threshold for the given set of parameters (1% performance level, 10,000 component system, 900 critical components, .06% weak components, A=3.9). The template is presented in a format independent manner.

The performance threshold is the probability one or more weak components remain.

Testcase shall be composed of all input data sets submitted for a single test run. Multiple runs may be performed within a testrun.

Failure behavior is affected by two principal factors:
the number of faults in the software being executed.
the execution environment or operational profile of execution.

Opportunities for faults are introduced when the code is created. A fault typically manifests itself through usage.

Code that is inherited does not usually introduce any appreciable number of faults, except at the interfaces.

The process of fault removal introduces the probability of new faults.

Based on the new code developed there are estimated to be 15,000 lines of new code associated with this test software

Worstcase there are 900 separate entities considered to be critical components. . The weak sub-population is a subset of this number.

The objective of the test software evaluation is to guarantee a threshold of performance. For this evaluation the performance threshold is set such that there was a 99% probability that the process would go to the field with no failures under the given assumptions. The assumptions were
the critical components was no more than 900
the weak sub-population was no more that .06%

Using those assumptions the Markovian burn-in graph appearing in Jensen/Petersen, page 107, yields the normalized value of 3.9.

The testcases shall be randomly drawn from the pool of available testcases. Testcases may be assembled into testruns so long as the random drawing order is maintained. Once drawn from the test pool, a testcase shall be withheld from further testing until the remainder of the test pool is exhausted or until a failure event occurs.

The independent variable shall be accumulated test cycles. A test cycle is composed of a submission of a testcase and the corresponding review of the differences to identify test anomalies. Test cycles are accumulate. Test cycles for testruns are accumulated in testcase order regardless of the order in which the review of differences takes place.

Test durations or periods shall be expressed in terms of test cycles.

The submission of a testcase to the test software may result in more than one failure event.

The submission of a testrun of the test software may result in only the first testcase member experiencing a failure event.

A failure event shall occur when an "un-resolved" difference has been determined. The test cycle, testcase identifier, expected parameter response, actual response parameter, and test operator shall be documented.

The failure event information shall be consolidate as test data. A Weibull plot shall be executed using the test data with a decision confidence limits Alpha/Beta set to 10/10 for test type of test to failure. The characteristic life for the testing and the Weibull plot shall be included with the consolidated failure information.

The test software shall not be released until it has demonstrated that it is a stable and mature. That condition is reached when the test software exhibits a failure free period such that normalized failure free period exceeds 3.9.

Definitions:

A: The normalization value of T_M / $MTTF_n$. This is used as the independent variable of the burn-in planning/analysis graphs.

$C_{N:n}$: The combination value. This value denotes the number of N distinct items taken n at a time.

$$C_{N:n} = {}_NC_n = \binom{N}{n} = \frac{N!}{(N-n)! \bullet n!}$$

exp(): The exponential function for the value given in the parenthesis.

fix: The attempt to resolve a software failure mechanism.

F_n: Probability of observing a number of n failures during a duration of test cases..

$$F_n = 1 - R_n$$

$F_{n.,n}$: Probability of no net change given that an n number of failures is observed.

$$F_{n.,n} = F_n \bullet [(1-p) \bullet P_{s,n} + p \bullet P_{wn}]$$

$F_{n.,n-1}$: Probability of an decrease of one prior failure given that an n number of failures is observed.

$$F_{n.,n+1} = F_n \bullet p \bullet P_{s,n}$$

$F_{n.,n+1}$: Probability of an increase of one new failure given that an n number of failures is observed.

$$F_{n.,n+1} = F_n \bullet (1-p) \bullet P_{wn}$$

M_{pop}: The main or strong population. This population exhibits a constant failure rate over an extended period of time. For our domain this is the population of software which is problem free from the beginning.

MTTF: Mean time to failure. For our solution domain this is the number of test cases execute prior to a failure.

N: The number of critical components. This group of components is a subset of the entire population of the components within a system. These software component are suspected as being potential contributors the bimodal distribution exhibited in a Weibull plot of the observed failures.

n: The number of weak components. This group is a subset of the critical components. From a Weibull plot, theses software components manifest themselves as those failures producing the characteristic "S" shape leading into main population failure segment. The value of n ranges from 0 to N.

$$n \subseteq N$$

p: The probability of making a bad fix. The complement of this value is (1-p) which represents the probability of making a good fix.

Population: The entire set of members of a unique grouping of items.

$$\text{Population} = M_{pop} + N$$

$P_{s,n}$: Probability that an observed failure was actually from the main population. By basic definition this term is valuede at zero for software.

$$P_{s,n} = 1 - (n \bullet \lambda_{weak}/\lambda_{main})$$

$P_{w,n}$: Probability that an observed failure was from the weak population.

$P_{w,n} = n \bullet \lambda_{weak}/\lambda_{main}$

q: The probability that an event will occur given that there are only two possible outcomes. The complement would be expressed as (1-q). For our domain this could be an expression of defects per line of code, defects per module, etc.

R_n: The probability that the system will complete testing without a failure.

$R_n = \exp(-\lambda_{system} \bullet T_M)$

T_M: The failure free period. For our solution domain this is the duration expressed in the number of test cases.

X!: The factorial value of X.

$X! = X \bullet (X-1) \bullet (X-2) \bullet \ldots \bullet (X-(X-1))$

Π_0: The initial probability vector. This vector displays the likelihood of 0 to N problems existing in the software just prior to entering the test sequence. This vector is actually composed of a set of probabilities. That set represents a collection of binomial probabilities.

$\Pi_0 = (\Pi_{(0)}, \Pi_{(1)}, \Pi_{(2)}, \ldots. \Pi_{(I)}, \ldots. \Pi_{(N)})$ where I ranges from 0 to N.

$\Pi_{(I)} = C_{N:n} \bullet q^n \bullet (1-q)^{N-n}$

Π_{RE}: The probability vector showing the likelihood of problems remaining after completion of testing. This vector is actually composed of a set of probabilities. That set represents the probability of 0,1,2..N problems remaining.

$\Pi_{RE} = (0_{RE}, 1_{RE}, 2_{RE}, \ldots I_{RE,}, \ldots N_{RE})$ where I ranges from 0 to N.

λ_{main}: The failure rate of the main population.

λ_{system}: The system failure rate.

$\lambda_{system} = n \bullet \lambda_{weak} + (N-n) \bullet \lambda_{main} = n \bullet (\lambda_{weak} - \lambda_{main}) + N \bullet \lambda_{main}$

λ_{weak}: The failure rate of the weak population.

$\subseteq$: The symbol that represents that a set A is a subset of set B. $A \subseteq B$

Equations and derivations:

$$F(x;\theta,\beta) = 1 - e^{-(x/\theta)^{\beta}}$$

β (Beta): Shape parameter or Weibull slope.

θ (theta): Scale parameter or the characteristic life (MTTF).

Equation 1: Weibull Failure Distribution

Rigorous mathematical presentation of the Weibull graph translation.

Starting with the cumulative distribution function for the Weibull:

$$F(t) = 1 - e^{-(t/\theta)^{\beta}}$$

$$F(t) - 1 = -e^{-(t/\theta)^{\beta}}$$

$$1 - F(t) = e^{-(t/\theta)^{\beta}}$$

$$\ln[1 - F(t)] = -(t/\theta)^{\beta}$$

$$-\ln[1 - F(t)] = (t/\theta)^{\beta}$$

$$\ln\left[\frac{1}{1 - F(t)}\right] = (t/\theta)^{\beta}$$

$$\ln\left[\ln \frac{1}{1 - F(t)}\right] = \beta \ln t - \beta \ln \theta = \beta[\ln t - \ln \theta]$$

$$\ln t = 1/\beta \ln\left[\ln \frac{1}{1 - F(t)}\right] + \ln \theta$$

on Weibull Paper

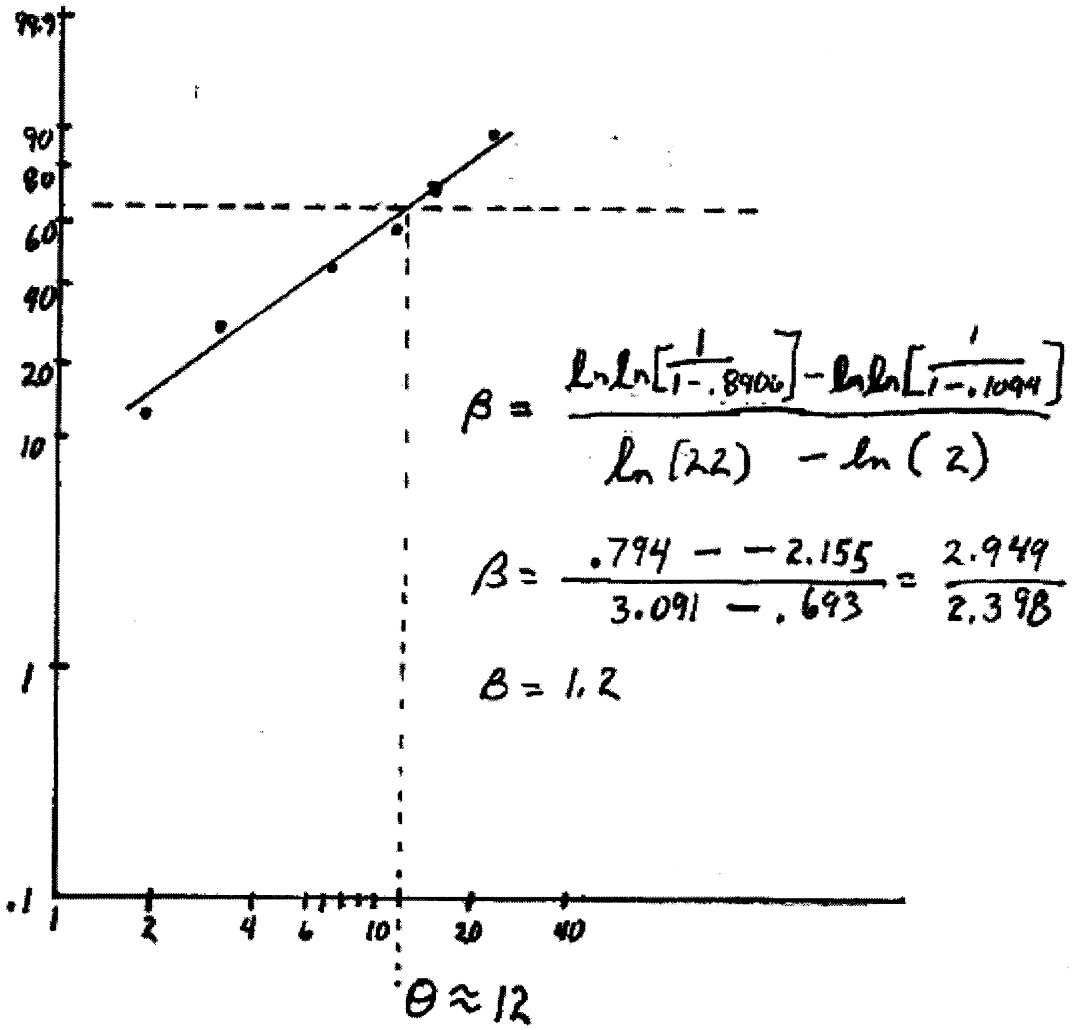
$$Y = \ln t \qquad X = \ln\left[\ln \frac{1}{1 - F(t)}\right] \qquad Y_{inc} = \ln \theta$$

and reversing the axis β = slope

$$Y = 1/\beta\, X + Y_{inc}$$

Equation 2: Deriving the slope y-intercept form of the Weibull function.

Determining Weibull Beta Slope between two points.



Equation 3: Determining Weibull Beta Slope between two points.

Rigorous Treatment of Markov Analysis:
This approach begins with an input matrix. The input matrix is multiplied by an intermediate matrix called a transfer matrix. The product of this matrix multiplication is an output matrix. The input matrix is actually the initial probability matrix based on the binomial distribution.

The transfer matrix is derived from manipulations of the state transitions matrix. The state transitions matrix composed of columns which indicate the existing states available after completion of a test case. The matrix rows represent the possible states of the software just prior the beginning of the test case. The R-columns and R-rows represents the probability of 0..N weak components within the software {after testing and before testing , respectively}. The F-columns and F-rows show the probabilities of incremental failure transition states within the software as resulting from the testing. Table 1. Probability of State Transitions shows a simple three component system. The simple system shown in Table 1 can have three or fewer failures.

State before test run	State after one test run								
		0_{RE}	**1_{RE}**	**2_{RE}**	**3_{RE}**	**0**	**1**	**2**	**3**
	0_{RE}	1							
	1_{RE}		1						
	2_{RE}			1					
	3_{RE}				1				
	0	R_0				$F_{0,0}$	$F_{0,1}$		
	1		R_1			$F_{1,0}$	$F_{1,1}$	$F_{1,2}$	
	2			R_2			$F_{2,1}$	$F_{2,2}$	$F_{2,3}$
	3				R_3			$F_{3,2}$	$F_{3,3}$

Table 1: Probability of State Transitions for three software faults leading to failure. Cells with "0" left blank for clarity.

This assignment decomposes the state transitions matrix into four sub-matrices. The R-columns to R-rows sub-matrix is the identity matrix, **I**.[9] The F-column to R-row matrix is a null matrix, **O**.[10] The R-columns to F-rows forms a diagonal matrix, **R**.[11] The **R** matrix displays the reliability vector along the principle diagonal.

The final sub-matrix, **Q**, evolves from an assumption that one of three things will happen as a result of a run of a test case. First, a test case run will result in no change in the number of problems. Second, the test case run may result in the introduction of new problems due to a faulty "fix". Finally, the test case run may detect a problem which is actually removed from the software. The probability of each of these three outcomes can be calculated.

With the state transition matrix established, the number of test attempts required before the software can complete a failure free period {test cases without a failure} of T_M can be calculated as $\mathbf{N}=(\mathbf{I}-\mathbf{Q})^{-1}$.[12] This leads to the calculation of the results matrix, $\mathbf{B}=\mathbf{N}\bullet\mathbf{R}$.[13] With the results matrix, **B**, and the initial probability vector both defined, Π_0, it is now possible to compute the probability of n-remaining problems, $\Pi_{RE}= \Pi_0\bullet\mathbf{B}$.[14]

Author: Tom Rooker
Tom has played a pivotal role in the transfer of knowledge within the regional Research Triangle Park, NCC software and reliability engineering community. His specific interests are the software engineering process and reliability engineering. He holds copyrights to a suite containing software and reliability engineering tools. The suite of tools can be obtained without charge from http://qed1.home.mindspring.com. His other published papers include works dealing with software development process models, Monte Carlo simulation, and warranty program planning. In his most recent role he was employed as a software product manager within the medical device arena. Previously he was employed in R&D developing of automated unit and component testing tools as well as graphical user interfaces. He has selectively provided his services as an independent contractor for over ten years. Prior to that he provided reliability engineering at Telex and Texas Instruments supporting computer communication products, large scale integration circuits, and laser guided weapon products. He is a licensed professional engineer and a certified reliability engineer. He can be contacted via qed1@mindspring.com.

---

[1] John Musa, Anthony Iannino, and Kazubira Okumto, Software Reliability, page 5.
[2] Ibid.
[3] Ibid.
[4] Petersen and Jensen, Burn-in, An Engineering Approach
[5] AAP012: Weibull Distribution Analysis, currently available free from Abbott Analytical Products, http://qed1.home.mindspring.com.
[6] AAP824: Software Test Planning /Analysis, currently available free from Abbott Analytical Products, http://qed1.home.mindspring.com.
[7] Petersen and Jensen,,pp. 158-160.
[8] Ibid, page 107.
[9] Standard Mathematical Tables, 16th Edition, page 109.
[10] Ibid.
[11] Ibid, page 107.
[12] Petersen and Jensen, Burn-in, An Engineering Approach, page 68.
[13] Ibid.
[14] Ibid., page 69.