A Moving Average Modeling Approach for Computing Component-Based Software Reliability Growth Trends

Wen-Li Wang[1], Thomas L. Hemminger[2], Mei-Huei Tang[3]
School of Engineering
[1, 2]Penn State University, Behrend College, Erie, PA 16563
wxw18@psu.edu[1], tlh5@psu.edu[2]
[3]Computer and Information Science Department
Gannon University, Erie,PA 16541
tang002@gannon.edu

Abstract. This paper introduces a moving average reliability growth model to describe the evolution of component-based software. In this model, the reliability of a system is a function of the reliabilities of its constituent components. The moving average provides a trend indicator to depict reliability growth movement within the evolution of a series of component enhancements. The moving average can reduce the effects of bias or measurement error of certain components by rendering a smoothed trend of system reliability growth. The input parameters are the components' configurations and individual reliability growths. The output is a vector of moving averaged system reliability growths indicating increasing component enhancement. The application of this model can facilitate cost/performance evaluation and support decision making for future software maintenance. More importantly, without introducing excessive computation, the model can be combined with many existing component-based reliability models to compute overall reliability growth.

Keywords: component-based software, moving average, convolution, fast Fourier transform, Markov model

(Received January 27, 2006 / Accepted May 19, 2006)

1. Introduction

Component-based software, consisting of a number of computational components, is known for easy adaptability, scalability, and reusability. It is believed to be the future trend in software development, and to have the same merit as in the hardware domain, e.g., telecommunications, electronics, and mechanics. It is characterized by exchanging system subsets of components, but not the whole, to increase efficiency and versatility.

Component-based software promotes reuse, interchangeability, and reliability. *Reuse* achieves high productivity. *Interchangeability* provides the flexibility of continuous updates, or upgrades, to software components, and makes software more adaptable to new technologies and environments. *Reliability* is a quality metric measuring the probability of error free operations of existing software systems.

Development technology exists for component-based software, including ActiveX, COM/DCOM, .NET, EJB, CORBA, OLE, OpenDoc, etc. Techniques have also been adopted and proposed to measure reliability. Cheung's user-oriented software reliability model [3] could address homogenous module interactions. In [5], the module-based approach was promoted to the component level. Krishnamurthy and Mathur [11] conducted an experiment to evaluate a CBRE method, estimating component execution path reliability for each test input and averaging the results. Gokhale et al [8] predicted architecture-based software reliability using measurements obtained from the regression test suite, and coverage measurements. In [7], a simulation approach was proposed. A general model [14] and the fundamental theory [9] for reliability measurement of component-based software were also available. In [20], we developed an architecture-based reliability model to address heterogeneous component interactions.

White-box is the common approach for these models due to its ability to address internal system structures, and to accommodate frequent component upgrades and updates. Although these models yield a reliability measure, they do not describe a trend of constant software improvement, and the noise introduced by improperly measured components may severely bias the result. Therefore, we develop a moving average (MA) reliability growth model to deal with these problems. Our approach extends the reliability representation from a single model to a model subspace. The MA, a useful and objective analysis tool, provides us with a trend. It is widely employed in a variety of domains, e.g., in business the MA has been used to smooth out fluctuations and diminish the impact of incorrect stock predictions.

The computation of MA reliability growth is a function of a series of discrete intervals. Each considers a

specific number of component enhancements in the execution paths without concern for any particular component. This is different from component sensitivity analysis [3], computed as $\partial R/\partial R_i$, the partial derivative of the system reliability R over the component reliability R_i. In fact, our model takes components configuration and reliability improvements as input, and outputs a vector of moving averaged system reliabilities corresponding to the discrete intervals. The MA approach integrates well with existing component-based reliability models, allowing them to provide system reliability growth trends and prevent bias from individual components. The outcome can assist in cost/performance evaluations and offers directions of improvement.

One goal of this paradigm is to prevent excessive computation. This is significant when integrating with existing reliability models, since most are computationally intensive and frequent component upgrades and updates often make re-computation of the entire model expensive and time-consuming. Therefore, in the MA approach, all component reliability improvements are entered simultaneously into a single matrix, and some entries, pertinent to a series of reliability improvements, are vectors rather than scalars. We utilize the fast Fourier transform (FFT) [2, 4] to perform convolutions [1] during certain matrix computations with the final result being a vector of MA reliability growths. The FFT is computationally efficient in performing vector multiplications, meaning that our approach provides solid performance in deriving trend of reliability growths, by reducing the computational load.

The rest of the paper is organized as follows. In section 2, we describe the convolution approach to computing the MA. In section 3 the FFT is adopted to improve performance. Section 4 introduces an algorithm incorporating the MA into the transition matrix. In section 5, we discuss integration with component-based software reliability models. Section 6 presents two applications of the MA reliability growth trend, and section 7 provides some concluding remarks.

2. Moving Average and Convolution

A MA [10] is a list of mean values over a specific set of discrete intervals. The value corresponding to an interval is the MA software reliability within that discrete interval. The next interval is designated for the reliability of an up-by-one component improvement in the execution paths. In other words, the *i*th interval computes the MA reliability, accounting for all execution paths, with each path having a number of *i* component enhancements, without regard for any specific components.

Figure 1 illustrates a system with two components c_x and c_y running in sequence. The reliability of component c_x, enhanced twice from *a* to *b* and then to *c*, is stored in a vector $f = [a\ b\ c]$. The reliability of c_y from *d* to *e* is in vector $g = [d\ e]$. Apparently, the system reliability varies from the beginning as *ad* to *ce*. However, *b* should not be neglected, because it plays an important role on the intermediate progression of reliability growth. Therefore, our model takes into account all evolving component enhancements to exhibit the growth trend. From vectors *f* and *g*, the number of component enhancements can be zero, one, two, or three. The reliability *ad* is the result of no improvement, while *ce* is the result of three enhancements with two on c_x and one on c_y. The reliability with one component improvement from *a* to *b* on c_x, and another from *d* to *e* on c_y is equal to (*bd*+*ae*)/2. Similarly, the reliability with two component improvements (one from *a* to *b* and to *c* on c_x, and the other from *a* to *b* on c_x and *d* to *e* on c_y) is (*cd*+*be*)/2. Consequently, the MA reliability growth shows the trend as [*ad* (*bd*+*ae*)/2 (*cd*+*be*)/2 *ce*] for an increasing number of component enhancements.

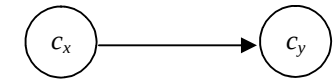

Figure 1: A sequential system with two components

In our model, the computation of a MA reliability growth trend takes advantage of convolution [1]. Convolution has been widely applied in a range of engineering applications, primarily to relate system inputs and outputs, and to perform correlation. It can also be used to compute the product of two polynomials. The convolution of two functions *f* and *g* is denoted by *f* * *g*. We utilize the discrete version of convolution as shown here:

$$y_n = f_n * g_n = \sum_{k=-\infty}^{\infty} f_k\, g_{n-k} \qquad (1)$$

The motivation of using convolution comes from its ability to compute the coefficients of the product of two polynomials. For example, given two vectors [*a b c*] and [*d e*] to represent polynomials $ax^2 + bx + c$ and $dx + e$, respectively, the convolution *f* * *g* in Eq. (1) yields [*ad bd*+*ae cd*+*be ce*]. In our approach, we store the reliability enhancements of software components in vectors and exploit convolution to determine reliability growth trends.

Convolution can relieve the computational burden, especially when encountering large vectors. Recall [*ad bd*+*ae cd*+*be ce*] is the convolution result of $f = [a\ b\ c]$ and $g = [d\ e]$. However, it has not yet met the expected MA [*ad* (*bd*+*ae*)/2 (*cd*+*be*)/2 *ce*], because the formula only sums up the values without performing an average. In other words, *f* * *g* also needs to be divided by the number of product terms in each discrete interval. Here, *ae*+*bd* is the sum of two product terms, which should be divided by 2. To compute the number of product terms in each interval two additional vectors are required $\overline{f}$ =

[1 1 1] and $\overline{g}$ = [1 1], having same lengths as f and g, respectively. The convolution $\overline{f} * \overline{g}$ yields [1 2 2 1]. After performing a component-wise division of $f * g$ by $\overline{f} * \overline{g}$, the outcome is the expected MA [ad (bd+ae)/2 (cd+be)/2 ce]. The following defines the MA reliability growth function $R(x,y)$ for two consecutive components c_x and c_y as:

$$R(x,y) = (f * g) ./ (\overline{f} * \overline{g}) \quad (2)$$

where "./" represents component-wise vector division.

3. Convolution and Fast Fourier Transform

MA software reliability growth for two consecutive components can be accomplished with two convolutions and a component-wise division, as in Eq. (2). Convolution requires $O(N^2)$ operations on two N-length vectors. Based on [2, 4], the fast Fourier transform (FFT) only requires $O(N \log_2 N)$ operations, turning convolutions into simple component-wise multiplications. This can significantly improve the computation efficiency for large-scale systems. The FFT is an efficient algorithm which is derived from the discrete Fourier transform (DFT). The DFT is a linear, invertible function [16, 17] of great importance to a wide variety of applications, such as processing signals, solving partial differential equations, and multiplying large integers, etc. The DFT transforms a vector of n complex numbers $x_0, \ldots, x_{n-1}$ into a vector of n complex numbers $f_0, \ldots, f_{n-1}$, presented as $F\{[x_0\ x_1 \ldots x_{n-1}]\} = [f_0\ f_1 \ldots f_{n-1}]$. The expression for the DFT is in Eq. (3), where $i = \sqrt{-1}$.

$$f_m = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i}{n} mk}, \; m = 0, \ldots, n\text{-}1 \quad (3)$$

The inverse DFT is presented as $F^{-1}\{[f_0\ f_1 \ldots f_{n-1}]\} = [x_0\ x_1 \ldots x_{n-1}]$ and is expressed by:

$$x_k = \sum_{m=0}^{n-1} f_m e^{\frac{2\pi i}{n} mk}, \; k = 0, \ldots, n\text{-}1 \quad (4)$$

An expression for the FFT is shown in Eq. (5). Based on a divide-and-conquer concept, a DFT of an even size n can be restated as the sum of two DFTs, each of size n/2 [2, 4]. One summation is from the even-numbered indices, and the other is from the odd-numbered indices. The procedure can be repeated recursively and is equivalent to Eq. (3). The FFT does require that vector lengths be a power of 2, but zeros can be added to those which do not meet this criterion with no effect on the result. This is known as *zero-padding*.

$$f_m = \sum_{k=0}^{\hat{n}-1} x_{2k} e^{-\frac{2\pi i}{\hat{n}} mk} + \sum_{k=0}^{\hat{n}-1} x_{2k+1} e^{-\frac{2\pi i}{\hat{n}} mk}$$

, where $\hat{n} = n/2$, $m = 0, \ldots, n\text{-}1$ (5)

In the following, we adopt the same notations F and F^{-1} to represent the FFT and its inverse. By [1, 2]:

$$f * g = F^{-1}\{F\{f\}.*F\{g\}\} \quad (6)$$

, where ".*" represents component-wise vector multiplication. From Eqs. (2) and (6), we derive Eq. (7) that utilizes FFT to compute the MA reliability growth $R(x,y)$ for two consecutive components c_x and c_y as:

$$R(x,y) = (f * g) ./ (\overline{f} * \overline{g})$$
$$= F^{-1}\{F\{f\} .* F\{g\}\} ./ F^{-1}\{F\{\overline{f}\} .* F\{\overline{g}\}\} \quad (7)$$

Eq. (7) can be extended to address a number of components $c_{i_1}, c_{i_2}, \ldots, c_{i_n}$ between c_x and c_y. In this case, $R(x,y)$ is computed as in Eq. (8).

$$R(x,y) = R(R(\ldots R(R(x,i_1), i_2), \ldots, i_n), y) \quad (8)$$

The following demonstrates the use of the FFT to compute the MA reliability growth function of Figure 1. Vectors f, g, $\overline{f}$, and $\overline{g}$ do not have lengths of 2^n, and are, therefore, zero-padded. Thus, we have f = [a b c 0], g = [d e 0 0], $\overline{f}$ = [1 1 1 0], and $\overline{g}$ = [1 1 0 0]. This enables the FFT to be employed to compute the component-wise multiplications and divisions. For illustrative purposes, we assign a, b, c, d, e the values of 0.9, 0.92, 0.95, 0.96, 0.98, respectively.

By Eq. (5),

$F\{f\}$ = [2.77 -0.05-0.92i 0.93 -0.05+0.92i]

$F\{g\}$ = [1.94 0.96-0.98i -0.02 0.96+0.98i]

$F\{\overline{f}\}$ = [3 -1i 1 1i]

$F\{\overline{g}\}$ = [2 1-1i 0 1+1i]

By Eq. (6), $f * g = F^{-1}\{F\{f\} .* F\{g\}\}$ = [0.864 1.7652 1.8136 0.931]

$\overline{f} * \overline{g} = F^{-1}\{F\{\overline{f}\} .* F\{\overline{g}\}\}$ = [1 2 2 1]

By Eq. (7),

$R(x,y) = (f * g) ./ (\overline{f} * \overline{g})$ = [0.864 1.7652 1.8136 0.931] ./ [1 2 2 1]

= [0.864 0.8826 0.9068 0.931]

The result from the FFT is identical to that of Eq. (2), which is verified by [ad (bd+ae)/2 (cd+be)/2 ce] = [0.864 0.8826 0.9068 0.931]. Obviously, this has a greater impact on larger problems.

4. Moving Average Reliability Modeling for Multiple Execution Paths

The next step is to model the entire system, addressing the configurations of software components in multiple execution paths. In general, system structures can be classified as follows:

1. A system has components running in one execution path.
2. A system has components with diverging execution paths.
    a. The number of component enhancements in each path is: identical or different.
    b. The number of execution paths is: finite or infinite.

Section 3 described a simple structure with two components in one execution path, with Eq. (8) addressing problems with one execution path, consisting of multiple components. Nevertheless, a software system is likely to contain multiple execution paths, with each path having a different number of component reliability enhancements. Moreover, the number of execution paths may not always be finite if the transitions among components form a cyclic loop. Consequently, computation of the MA for different paths is likely to require distinct vector lengths, resulting in difficulty when computing the MA for the entire system. An infinite number of execution paths also emphasizes the challenge of modeling system MA reliability growth, because each discrete interval now has to consider an infinite number of paths for modeling increased component enhancement.

In section 4.1, we discuss MA modeling for a system with a finite number of execution paths and introduce an algorithm to standardize vector lengths. In section 4.2, the foundation and algorithms for modeling an infinite number of execution paths through a transition matrix are introduced.

4.1. Moving Average Measurement with Different Vector Sizes

The number of component enhancements in each execution path can be distinctive, resulting in different vector lengths, as illustrated in Figure 2. As shown, the first execution path is $c_1 \rightarrow c_2 \rightarrow c_4$ with probability p and the second is $c_1 \rightarrow c_3 \rightarrow c_4$ with probability $1–p$. The first path has a single component enhancement in component c_1 from a to b. The second path has two component enhancements, one is in component c_1 from a to b, and the other in component c_3 from d to e.

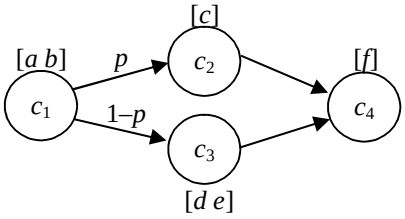

Figure 2: A system with two execution paths

From Eq. (8), the first path of $R(1,4)$, enumerated as $R(1,4)_1$, equals $R(R(1,2),4)$ = [*acf bcf*]. The second path, $R(1,4)_2 = R(R(1,3),4)$ = [*adf* (*ae*+*bd*)*f*/2 *bef*]. This presents a problem. The result cannot be computed as $p * R(1,4)_1 + (1-p) * R(1,4)_2$, because the vector lengths derived from these two paths are not identical. Therefore, the computation of $R(1,4)$, a combination of $R(1,4)_1$ and $R(1,4)_2$, requires adjustment of the short vector $R(1,4)_1$ to be the same length as the longer vector $R(1,4)_2$.

Padding of the shorter vector is necessary to match the length of $R(1,4)_2$. This ensures size consistency, and maintains the reliability growth of the original short vector. We call this procedure an *equalization process*, which is conceptually similar to zero-padding mentioned above. For example, the short vector [*acf bcf*] of $R(1,4)_1$ will be padded with the last vector element yielding [*acf bcf bcf*]. Padding does not affect the reliability growth of the first path, which remains unchanged as *bcf*. The following presents the algorithm *equalize* which facilitates the equalization process. The algorithm requires two input vectors, with the shorter vector forced to be the same length as the longer one. Calling the function *equalize*($R(1,4)_1$, $R(1,4)_2$) yields two vectors [*acf bcf bcf*] and [*adf* (*ae*+*bd*)*f*/2 *bef*] having the same length. Therefore, $R(1,4)$ can be computed as p * [*acf bcf bcf*] + (1–p) * [*adf* (*ae*+*bd*)*f*/2 *bef*].

```
// Equalize the size of A and B vectors
equalize(Vector A, Vector B) {
    if (A→size == B→size)
        return;
    else if (A→size < B→size) {
    // equalizing to the size of A to be the same as B
        for (i = A→size+1; i <= B→size; i++) {
            A(i) = A(i-1);
        }
    }
    else {
    // equalizing to the size of B to be the same as A
        for (i = B→size+1; i <= A→size; i++) {
            B(i) = B(i-1);
        }
    }
}
```

As a consequence, the formula for $R(x,y)$, with n execution paths between components c_x and c_y, can be generalized as in Eq. (9), noting that *equalize* is always called beforehand.

$$R(x,y) = \sum_{k=1}^{n} p_k * R(x,y)_k$$

$$\text{, where } n > 1,\ 0 < p_k \le 1, \text{ and } \sum_{k=1}^{n} p_k = 1 \qquad (9)$$

4.2. Model Foundation for an Infinite Number of Execution Paths

Here we discuss the foundation of our MA model, which will aid in its integration with component-based software reliability models that take into account an infinite number of execution paths. Eq. (9) can be applied to deal with a finite number of execution paths without cyclic loops, e.g., integration with our web-based reliability model [19]. However, the transitions among components often form cyclic loops, forcing the number of execution paths to be infinite, thus requiring an alternate solution.

For this situation a number of paradigms [3, 6, 12, 15] have been proposed to construct a transition matrix and exploit Markov models for reliability computations. The transition matrix considers the reliabilities of, and the interrelationships among, software components. Construction of the transition matrix can follow existing component-based reliability models [3, 8, 18]. The major difference is that the entries of our matrix are not necessarilly scalar values. Some entries, corresponding to a sequence of component upgrades or updates, become hidden vectors representing individual components' reliability enhancements. The elements in a vector need not be monotonic and may fluctuate because component upgrades and updates do not guarrantee an increasing reliability improvement, but may sometimes degrade the reliability by introducing additional faults. Once construction of the transition matrix is complete, we employ the reliability formulas of the component-based reliability models to compute MA software reliability.

Nevertheless, reliability formulas cannot be directly used to manage the combination of scalars and vectors within the transition matrix. There are certain difficulties, including the handling of computations between scalars and vectors, and the resolution of vector length. This is a significant challenge because many reliability formulas need to compute the determinant of the transition matrix, but there is no method for computing a vector of MA determinants for such a matrix. Therefore, we introduce another algorithm which takes advantage of the equalization process to address differing vector lengths. Convolution of these vectors follows the aforesaid simple vector-to-vector component-wise multiplication via the FFT.

The following illustrates the computation of MA determinants. Given three transition matrices M_2, M_3 and M_n, where M_2 is 2×2, M_3 is a 3×3, and M_n is $n \times n$. The elements can be scalars or vectors, and we denote a_{ij} to be the entry of a transition matrix in the ith row and jth column.

$$M_2 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, M_3 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix},$$

$$M_n = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

For our model, the MA determinant of M_2, denoted as $|M_2|$, is no longer computed as $a_{11}a_{22} - a_{12}a_{21}$. Based on Eq. (7), the value of $|M_2|$ can now be a vector as shown in Eq. (10), after application of the equalization process $equalize(R(a_{11}, a_{22}), R(a_{12}, a_{21}))$.

$$|M_2| = R(a_{11}, a_{22}) - R(a_{12}, a_{21}) \quad (10)$$

Computation of $|M_3|$ is an extension to that of $|M_2|$. Equalization is performed when encounting a vector computation such as "+", "–", ".*", and "./". For a scalar matrix, $|M_3|$ was originally computed as:

$$a_{11}\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} - a_{12}\begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} + a_{13}\begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

Evaluation of $|M_3|$ takes advantage of Eq. (10) and results in:

$$\begin{aligned} |M_3| = {} & R(a_{11}, R(a_{22}, a_{33}) - R(a_{23}, a_{32})) - \\ & R(a_{12}, R(a_{21}, a_{33}) - R(a_{23}, a_{31})) + \\ & R(a_{13}, R(a_{21}, a_{32}) - R(a_{22}, a_{31})) \end{aligned} \quad (11)$$

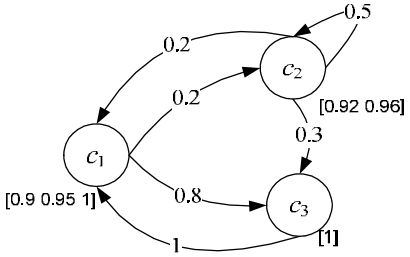

Figure 3: An example for computing $|M_3|$

Figure 3 shows a system consisting of three components c_1, c_2, and c_3 to demonstrate computation of $|M_3|$. Each entry, $M_3(i,j)$, stores the transition probability that c_i is fault free and transits to c_j. Let c_1 have two reliability enhancements stored as [0.9 0.95 1], c_2 have one enhancement [0.92 0.96], and c_3 have no enhancement represented by [1]. From the diagram, the transition probability from c_1 to c_2 is 0.2, and 0.8 to c_3. The transition probability from c_2 to c_1 is 0.2, 0.5 to c_2, and 0.3 to c_3. c_3 transits to c_1 with a probability 1. As a result, M_3 is constructed as follows:

$$\begin{bmatrix} 0 & .2\times[.9\ .95\ 1] & .8\times[.9\ .95\ 1] \\ .2\times[.92\ .96] & .5\times[.92\ .96] & .3\times[.92\ .96] \\ 1\times[1] & 0 & 0 \end{bmatrix}$$

By Eq. (11), $|M_3|$ =

$R(0, R([.46\ .48], 0) - R([.276\ .288], 0)) -$
$R([.18\ .19\ .2], R([.184\ .192], 0) - R([.276\ .288], 1)) +$
$R([.72\ .76\ .8], R([.184\ .192], 0) - R([.46\ .48], 1))$

And from Eq. (7), we have $R(x, y)$ =

$F^{-1}\{F(f)$.* $F\{g\}\}$ ./ $F^{-1}\{F\{\overline{f}\}$.* $F\{\overline{g}\}\}$.

Thus,

R([.46 .48], 0) = [0 0]

R([.276 .288], 0) = [0 0]

R([.184 .192], 0) = [0 0]

R([.276 .288], 1) = [.276 .288]

R([.46 .48], 1) = [.46 .48]

Accordingly, $|M_3|$ becomes:

R(0, [0 0]) –

R([.18 .19 .2], [–.276 –.288]) +

R([.72 .76 .8], [–.46 –.48]) =

[0 0] – [–0.0497 –0.0521 –0.0550 –0.0576] +

[–0.3312 –0.3476 –0.3664 –0.3840] =

[–0.2815 –0.2955 –0.3114 –0.3264]

Note, the equalization process is applied to the vector computations so that [0 0] is equalized to [0 0 0 0] before including the 4-element vectors.

The result can be verified through standard matrix calculations. The first element requires no component enhancements. Thus, we have:

$$\left\| \begin{bmatrix} 0 & .2\times.9 & .8\times.9 \\ .2\times.92 & .5\times.92 & .3\times.92 \\ 1 & 0 & 0 \end{bmatrix} \right\| = -0.2815$$

The second element requires one component enhancement, which can be one occurance of c_1 or c_2 as shown below:

$$\left(\left\| \begin{bmatrix} 0 & .2\times.9 & .8\times.9 \\ .2\times.92 & .5\times.92 & .3\times.92 \\ 1 & 0 & 0 \end{bmatrix} \right\| + \left\| \begin{bmatrix} 0 & .2\times.9 & .8\times.9 \\ .2\times.92 & .5\times.92 & .3\times.92 \\ 1 & 0 & 0 \end{bmatrix} \right\| \right) / 2$$

= -0.2955

The third element needs two component enhancements, c_1 twice, or both c_1 and c_2 once as follows:

$$\left(\left\| \begin{bmatrix} 0 & .2\times1 & .8\times1 \\ .2\times.92 & .5\times.92 & .3\times.92 \\ 1 & 0 & 0 \end{bmatrix} \right\| + \left\| \begin{bmatrix} 0 & .2\times.95 & .8\times.95 \\ .2\times.96 & .5\times.96 & .3\times.96 \\ 1 & 0 & 0 \end{bmatrix} \right\| \right) / 2$$

= -0.3114

The fourth element has three component enhancements with c_1 twice and c_2 once and is computed as:

$$\left\| \begin{bmatrix} 0 & .2\times1 & .8\times1 \\ .2\times.96 & .5\times.96 & .3\times.96 \\ 1 & 0 & 0 \end{bmatrix} \right\| = -0.3264$$

It is clear that our MA approach significantly reduces the computational complexity in comparison to conventional methods. In the following, we introduce an algorithm, based on Eqs. (10) and (11), to generalize computation of $|M_n|$, the MA determinant of a $n \times n$ matrix M, where $n \geq 1$. When $n = 1$ the algorithm returns a_{11}, but when $n = 2$ it returns a result from Eq. (10). When $n \geq 3$, it functions recursively to compute $|M_n|$ as $\sum_{i=1}^{n} R(a_{i1}, \text{cofactor}(M_n, i, 1))$. The cofactor($M_n$, i, 1) returns $(-1)^{i+1} \times |M_{n-1}|$, where M_{n-1} is a submatrix by removing the ith row and the first column of M_n. The recursion does not stop until the size of the submatrix is reduced to 2 by 2.

```
// M is a square mixtured matrix
Vector determinant(Matrix M)
{
    // a_xy is the entry value of M(x,y)
    if (M→row == 1)
        return a_11;
    else if (M→row == 2) {
        equalize(R(a_11, a_22), R(a_12, a_21));
        return R(a_11, a_22) – R(a_12, a_21);      // |M_2|
    }
    else {
         Vector vector = []; // to initialize
          for (i = 1; i <= M→row; i++)
              if (a_i1 != 0) {
                 equalize(vector, R(a_i1, cofactor(i, 1));
                 vector += R(a_i1, cofactor(M, i, 1));
              }
          return vector;
     }
}

Vector cofactor(Matrix M, int x, int y)
{
    // SM is a submatrix of M
    Matrix SM = new Matrix(M→row–1, M→column–1);
    for (i = k = 1; k <= M→row; k++) {
        if (k != x) {
            for (j = l = 1; l <= M→column; l++)
                if (l != y) {
                    // a_kl is the entry value of M(k,l)
                    SM[i, j] = a_kl;
                     j++;
                }
             i++;
        }
     }
     return (–1)^(x+1) * (–1)^(y+1) * determinant(SM);
}
```

5. An Example of Model Integration

This section illustrates an example of the integration of our MA model with Cheung's user-oriented software reliability paradigm [3]. This example serves as a guide-line for the integration of our MA approach with other component-based software reliability models. Cheung took into account branching and cyclic-loop structures,

being able to address an infinite number of execution paths. A formula was developed for computing software reliability R as shown in Eq. (12). M is an $n \times n$ transition matrix and I is an $n \times n$ identity matrix. $|I–M|$ is the determinant of matrix $(I–M)$. $|(I–M)_{n,1}|$ is the determinant of the submatrix, excluding the last row and first column of matrix $(I–M)$.

$$R = (-1)^{n+1} \frac{\left|(I-M)_{n,1}\right|}{\left|I-M\right|} R_n \tag{12}$$

Let a_{ij} be the entry value of M's ith row and jth column. The entries are calculated based on Eq. (13). R_i is the reliability of component c_i and P_{ij} is the transition probability from c_i to c_j.

$$M(i,j) = \begin{cases} a_{ij} = R_i P_{ij}, P_{ij} \neq 0 \text{ and } i \neq j \\ a_{ij} = 0, \text{otherwise} \end{cases}, \text{ for } 1 \leq i, j \leq n \tag{13}$$

Consider Figure 4, where R_i of c_i and P_{ij} between c_i and c_j are shown in Table 1. The $I–M$ matrix is constructed as shown in Figure 5. By Eq. (12), the reliability R is calculated as 0.8355. If R_1, R_2, …, R_{13} are all equal to 1.0, the reliability R is equal to 1.0.

Let R_1, R_2, …, R_{13} be replaced with values in Table 2 that contain reliability growths for different components. We fill the matrix with these vectors and scalars, and utilize our previous determinant algorithm to compute $|I–M|$ and $|(I–M)_{13,1}|$. Accordingly, the MA reliability growth trend R is computed:

$|I–M|$ = [0.73397 0.7343 0.72968 0.72814 0.72644 0.72542 0.72468 0.72376 0.72273 0.72135 0.72037 0.71895 0.71831 0.71681 0.71568 0.71305 0.7074 0.7074 0.7074]

$|(I–M)_{13,1}|$ = [0.61323 0.61448 0.62323 0.63086 0.639 0.64617 0.65269 0.65901 0.66481 0.66999 0.67457 0.67815 0.6809 0.68357 0.6866 0.69144 0.69489 0.70579 0.7074]

$$R = (-1)^{n+1} \frac{\left|(I-M)_{n,1}\right|}{\left|I-M\right|} R_n = (-1)^{14} \frac{\left|(I-M)_{13,1}\right|}{\left|I-M\right|} R_{13}$$

= [0.8355 0.83682 0.85411 0.86641 0.87964 0.89074 0.90066 0.91054 0.91986 0.9288 0.93642 0.94324 0.94792 0.95363 0.95937 0.96969 0.98231 0.99773 1.0]

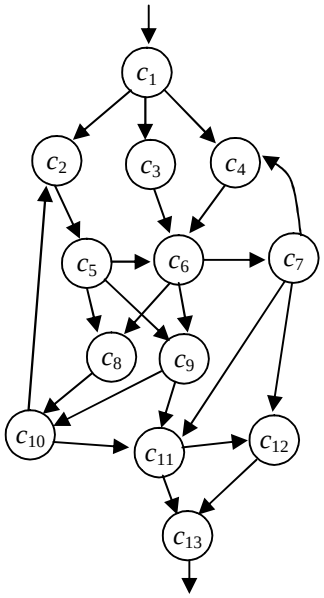

Figure 4: The state machine of a sample system with 13 components

$R_1 = 0.98$	$R_2 = 0.96$	$R_3 = 0.956$
$R_4 = 1.0$	$R_5 = 0.98$	$R_6 = 0.99$
$R_7 = 0.94$	$R_8 = 0.95$	$R_9 = 1.0$
$R_{10} = 1.0$	$R_{11} = 0.96$	$R_{12} = 0.98$
$R_{13} = 1.0$		
$P_{1,2} = 0.5$	$P_{1,3} = 0.2$	$P_{1,4} = 0.3$
$P_{2,5} = 1.0$		
$P_{3,6} = 1.0$		
$P_{4,6} = 1.0$		
$P_{5,6} = 0.4$	$P_{5,8} = 0.3$	$P_{5,9} = 0.3$
$P_{6,7} = 0.3$	$P_{6,8} = 0.4$	$P_{6,9} = 0.3$
$P_{7,4} = 0.1$	$P_{7,11} = 0.4$	$P_{7,12} = 0.5$
$P_{8,10} = 1.0$		
$P_{9,10} = 0.5$	$P_{9,11} = 0.5$	
$P_{10,2} = 0.4$	$P_{10,11} = 0.6$	
$P_{11,12} = 0.5$	$P_{11,13} = 0.5$	
$P_{12,13} = 1.0$		

Table 1: The reliability and transition probabilities of Figure 4

R_1 = [0.98 0.975 1.0],	R_2 = [0.96 0.957 1.0],	R_3 = [0.956 1.0]
R_4 = 1.0,	R_5 = [0.98 0.9726 1.0],	R_6 = [0.99 0.98 1.0]
R_7 = [0.94 0.96 1.0],	R_8 = [0.95 0.983 0.97 1.0],	R_9 = 1.0
R_{10} = 1.0,	R_{11} = [0.96 1.0],	R_{12} = [0.98 0.976 0.972 1.0]
R_{13} = 1.0		

Table 2: Individual component reliability growths through upgrades or updates

$$I\text{–}M = \begin{bmatrix}
1 & -.5R_1 & -.2R_1 & -.3R_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & -R_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -R_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & -R_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -.4R_5 & 0 & -.3R_5 & -.3R_5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -.3R_6 & -.4R_6 & -.3R_6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -.1R_7 & 0 & 0 & 1 & 0 & 0 & 0 & -.4R_7 & -.5R_7 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -R_8 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -.5R_9 & -.5R_9 & 0 & 0 \\
0 & -.4R_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -.6R_{10} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -.5R_{11} & -.5R_{11} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -R_{12} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Figure 5: The *I-M* matrix for the system in Figure 4

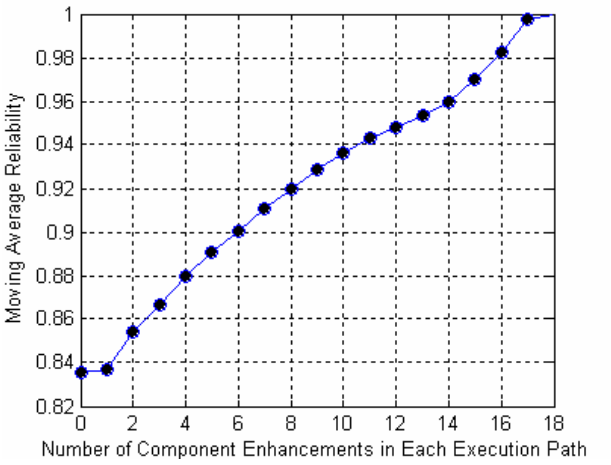

Figure 6: Software Reliability Growth Trend

Figure 6 shows the trend from vector *R*. It is clear that the first and last elements are 0.8355 and 1.0 respectively, and identical to the results of the previous pure scalar transition matrices. The intermediate discrete intervals concern only the number of enhancements, without considering a specific component; therefore, it is less biased. The vertical axis represents the average reliability estimate, and the horizontal axis lists the number of enhancements. In other words, *k* component enhancements means that each execution path has a MA reliability of exactly *k* upgrades or updates, and the system reliability is the sum of the MA reliabilities of all execution paths multiplied by the probability of traversing each path. Our algorithm in section 4 makes the computation of MA reliability growth trends feasible for models that can address an infinite number of execution paths.

6. Two Applications of the Moving Average Model

We have demonstrated the integration of our MA technique with Cheung's user-oriented software reliability model. The same modeling approach can be applied to other component-based software reliability models. For instance, integration with an architecture-based reliability model [18] will depict the reliability growth trend for software with heterogeneous architectures. Our MA model can be applied in many areas, e.g., in the following, we discuss two application domains, one for cost/performance evaluation and the other for software maintenance decision making.

6.1. Cost/Performance Evaluation

To remain competitive, a business may need to decide the cost, including man-hours and budget, to enhance the quality of software. Cost is always a factor regardless of whether the enhancement is to improve existing on-hand components or to purchase new COTS components. The highest performance for the lowest cost is a typical objective. With this in mind, let cost be a function of the number of component enhancements. A steep rise of this function indicates a high cost to include extra enhancements and/or provide upgrades. If cost/performance can be evaluated, it will guide software maintenance schedule. With our MA model, we define a cost/performance efficiency function to serve this purpose, as follows:

$$E(i,j) = \frac{Cost(j) - Cost(i)}{(R_j - R_i) * (j - i)},\ i < j \qquad (14)$$

$E(i,j)$ computes an average cost per unit reliability improvement, in the interval between *i* and *j* component enhancements. $Cost(i)$ and $Cost(j)$ are two accumulative costs for a number of *i* and *j* component improvements, respectively. R_i and R_j are the MA reliabilities for *i* and *j* component enhancements. Since $Cost(j) > Cost(i)$, $E(i,j)$ is positive when $R_i < R_j$; otherwise $R_i > R_j$. If $R_i = R_j$, $E(i,j)$ is assigned 0 due to no reliability improvements.

When $E(i,j)$ is positive, the lower the value, the better the cost/performance ratio. A negative value of $E(i,j)$, caused by declining reliability, states the opposite, i.e., greater negative values indicate poorer performance.

Here we show an example based on the MA results from Figure 6. Assume a company spent 500 man-hours to make 12 component improvements. For 600 man-hours it observed 14 improvements. After the company invested 1100 man-hours, 18 component improvements were reported. We understand that the MA software reliabilities for 12, 14, and 18 component reliability improvements are 0.94792, 0.95937, and 1.0, respectively. Accordingly, $E(12,14)$ and $E(14,18)$ can be computed as follows:

$$E(12,14) = \frac{600-500}{(.95937-.94792)(14-12)} = 4367$$

$$E(14,18) = \frac{1100-600}{(1-.95937)(18-14)} = 3077$$

Since $0 < E(14,18) < E(12,14)$, the results indicate a better cost/performance in the interval between 14 and 18 component enhancements. In this region, the average cost for each component enhancement is 125 man-hours, much higher than 50 as in the segment between 12 and 14. Yet the region between 14 and 18 experiences a significant reliability improvement enhancement over that of 12 and 14, i.e., (1-0.95937)/4 = 0.0101575 versus (0.95937-0.94792)/2 = 0.005725 respectively. By taking both cost and reliability improvement into account, the cost/performance ratio has a more compelling result between the segment 14 and 18.

6.2. Decision Making

The MA reliability growth model can also support decision making. One important decision may be to determine whether or not to continue further reliability improvement. This requires consideration of certain attributes, such as deadline, budget, man-hours, and the effectiveness of the improvements. A significant improvement may make the investment much more attractive. One advantage of our MA reliability growth model is its ability to depict the trend of reliability growth. The trend is a key indicator for the system as a whole, not just a single component, because the approach smoothes out flunctuations and prevents potential bias from individual components. In other words, the model eliminates bias while providing a good confidence level for judging the effectiveness of future system reliability improvements.

Figure 7 compares two MA reliability growth trends to study which situation has better potential for future reliability improvement. Both trends start with reliability 0.8355 and end at 0.9422. The upper trend shows an instant escalation then slows down, while the lower trend is nearly linear. It is clear that the upper trend starts to show a convergence to a horizontal assyptote. This suggests that an additional component enhancement to the lower system yields a more significant reliability improvement than the other. In summary, a trend that depicts a fairly linear growth is a stronger candidate for further improvement.

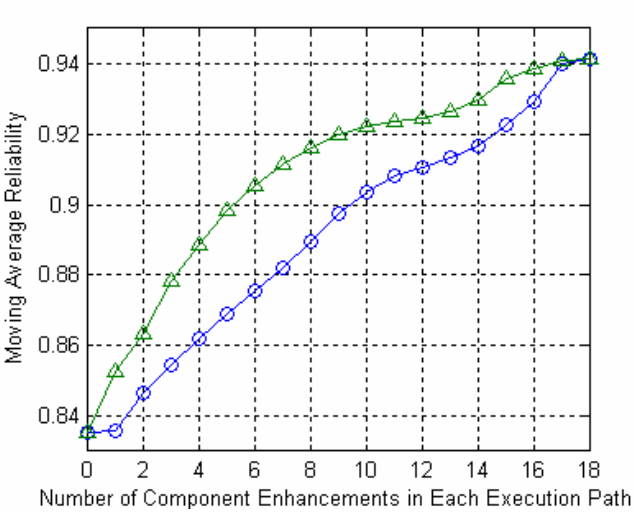

Figure 7: Two Software Reliability Growth Trends

There is also a possibility to observe a trend through the vectors of individual components, but the best way is to observe the entire trend. This is particularly the case when the vectors show a variety of unstable fluctuations, making individual observations difficult to interpret.

7. Conclusion

We have developed an MA software reliability growth model for component-based software. This model depicts a growth trend of software reliability instead of a single reliability measure. The trend is an averaged function of an increasing number of component enhancements, regardless of the characteristics of any specific constituent components. The computation of MA is through convolutions, which smooth out sudden impulses to reveal a trend. Consequently, the reliability is less biased by the incorrect measure of a single component, or a small set of components. The computation of convolutions can be time consuming, so the fast Fourier transform is employed to improve performance allowing for the analysis of a large-scale software system. Our model addresses the number of component enhancements in individual execution paths and integrates them to compute the MA. The enhancements for each component are stored as a vector. By embedding all vectors into a transition matrix, our proposed algorithm for the model is capable of addressing a finite as well as an infinite number of execution paths, making it extremely versatile.

Our MA approach can be easily merged with existing white-box based software reliability models, permitting engineers in their own fields to adopt and adapt the technique to meet specific needs. A complete example demonstrates the integration. Two applications of this model were discussed to facilitate cost/performance evaluations and support decision making. We define a cost/performance ratio metric to help evaluate the effectiveness of reliability improvement, and discuss the trend patterns to continue or stop the improvement process. This work is expected to be applicable to solve many other, more complex, problems.

References

[1] Arfken, G., "Convolution Theorem." in *Mathematical Methods for Physicists, 3rd ed.*, pp. 810-814, Orlando, FL: Academic Press, USA, 1985.

[2] Brigham, E. O., *The Fast Fourier Transform and Its Applications*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1988.

[3] Cheung, R. C., "A User-Oriented Software Reliability Model", In *IEEE Transactions on Software Engineering*, 6(2):118, pp.118-125, March, 1980.

[4] Cooley, J. W. and Tukey, J. W., "An Algorithm for the Machine Calculation of Complex Fourier Series," In *Mathematics of Computation*, 19(90), pp.297–301, 1965.

[5] Dolbec, J. and Shepard, T., "A Component Based Software Reliability Model", In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*, pp. 19-28, Toronto, Ontario, Canada, 7-9 November, 1995.

[6] Goel, A. L. and Okumoto, K., "A Markovian Model for Reliability and Other Performance Measures of Software Systems", In *Proceedings of National Computer Conference*, pp.769-775, New York, NY, USA, June, 1979.

[7] Gokhale S. S., Lyu M. R., and Trivedi K. S., "Reliability Simulation of Component-Based Software Systems", In *Proceedings of 9th International Symposium on Software Reliability Engineering (ISSRE)*, pp192-201, Paderborn, Germany, 4-7 November, 1998.

[8] Gokhale, S. S., Wong, W. E., Trivedi, K. S., and Horgan, J. R., "An Analytical Approach to Architecture-Based Software Reliability Prediction", In *Proceedings of IEEE International Computer Performance and Dependability Symposium (IPDS)*, pp.13-22, Durham, NC, USA, 7-9 September, 1998.

[9] Hamlet D., Mason D., and Woit D., "Theory of Software Reliability Based on Components", In *Proceedings of 23rd International Conference on Software Engineering (ICSE)*, pp361-370, Toronto, Ontario, Canada, May 12-19, 2001.

[10] Kenney, J. F. and Keeping, E. S., "Moving Averages", *Mathematics of Statistics*, Pt. 1, 3rd ed., pp. 221-223, Van Nostrand, Princeton, NJ, USA, 1962.

[11] Krishnamurthy, S. and Mathur, A. P., "On the Estimation of Reliability of a Software System Using Reliabilities of its Components", In *Proceedings of 8th International Symposium on Software Reliability Engineering (ISSRE)*, pp.146-155, Albuquerque, NM, USA, 2-5 November, 1997.

[12] Littlewood, B., "A Reliability Model for Systems with Markov Structure", *Applied Statistics*, 24(2), pp.172-177, February, 1975.

[13] Lo, J-H, Huang, C-Y., Kuo, S-Y. and Lyu, M.R., "Sensitivity Analysis of Software Reliability for Component-Based Software Applications", In *Proceedings of 27th International Computer Software and Applications Conference*, pp. 500-505, Dallas, Texas, USA, 3-6 November, 2003.

[14] Mao, X. and Deng, Y., "A General Model for Component-Based Software Reliability", In *Proceedings of 29th Euromicro Conference*, pp. 395-398, Antalya, Turkey, 1-6 September, 2003.

[15] Musa, J., Iannino, A., and Okumoto, K., *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, NY, USA, 1987.

[16] Oppenheim, A. V., Schafer, R. W., and Buck, J. R., *Discrete-Time Signal Processing*, Prentice-Hall, 1999.

[17] Smith, S. W., *The Scientist and Engineer's Guide to Digital Signal Processing, 2nd ed.*, California Technical Publishing, San Diego, CA, USA, 1999.

[18] Wang, W. L., Pan, D. and Chen, M. H., "Architecture-Based Software Reliability Modeling", *Journal of Systems and Software*, 79(1), pp 132-146, January, 2006.

[19] Wang, W. L. and Tang, M., "User-Oriented Reliability Modeling for a Web System". In *Proceedings of 14th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 293-304, Denver, Colorado, 17-20 November, 2003.

[20] Wang, W. L., Wu, Y. and Chen, M. H., "An Architecture-Based Software Reliability Model", In *Proceedings of Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp.143-150, Hong Kong, China, 16-17 December, 1999.