An Approach for Identifying Failure-Prone State of Computer System

Yun-Fei Jia and Renbiao Wu
Tianjin Key Laboratory for Advanced Signal Processing, Civil Aviation University of China, China

Abstract: *Controlled experiment can help us to better understand the root origin and evolution of software aging. Detection and/or quantification of software aging is an important research issue. The experimental observations may be obscure, although it may implicate much useful information. In this paper, we first report the memory thrashing phenomenon observed in our controlled experiment, and find the vibration frequency of available memory may be a potential indicator of aging. We then characterize and measure the vibration frequency by using amplitude spectrum analysis. Accordingly, a metric is proposed to measure the aging extent implicated in the vibration frequency by using power spectrum analysis. Finally, we propose an approach for online aging detection based on sliding window Fourier transform. The metric is calculated for each "window" to evaluate the severity of aging at a given time instant.*

Keywords: *Software aging, software maintenance, fourier analysis.*

Received March 20, 2014; accepted February 10, 2015

1. Introduction

It is considered the root causes of software aging come from residual defects in the software system [3]. For example, when an application server executes continuously long period of time, it may accumulates many error conditions in the process space and/or kernel space, such as memory leak, round off error, unreleased file lock, etc. Consequently, the application server will show lower performance gradually. In other words, software aging results from degradation of runtime environment. Hence, the subject investigated in software aging refers to the whole computer system, including operating system and all applications running on it.

Software aging indicators are an important research issue, because they can point out when the software system ages, and how "old" the software system is. Aging indicators can be extracted from resource usage and/or performance indicator. For example, software aging can be detected by monitoring the activity of computer systems. When aging becomes more and more serious, one and/or more resource variable may become a bottleneck for performance of computer system. This type of "risky condition" can be used to forecast the onset of aging. Identifying the risky condition before serious failure or crash is an issue of paramount importance.

Empirical study can help us better understand the root origin of software aging based on collected aging-related data, such as available memory, swap space usage, network throughout, response time, etc. Nevertheless, if the data are incorrectly collected or the collected data are not repeatable, the resulting analysis may mis-validated. Usually, empirical studies includes experimental researches and engineering researches. In this paper, controlled experiment refers to those experiments, in which, experimental conditions can be specified by human, including configuration of subject software and operating system, etc. Hence, the experimental results can be repeatable, and the resulting theory can lead engineering practice reasonably.

Researchers have proposed many assumptions on its mechanism, but the nature of software aging is not completely studied and/or well understood because of lack of massive controlled experiments. Although several experimental studies are reported, they are not the majority of software aging researches. So far, less than twenty publications discussing experimental studies on software aging can be found on major software and reliability journals [3, 7, 10]. This contrasts with the growing awareness and widely accepted importance of experiment-based studies [1]. This is because software experiments are usually time-consuming, workload-intensive and error-prone.

Due to lack of reported experimental studies on software aging, rare metrics have been proposed to measure this phenomenon [3, 7, 9]. Those metrics aim to measure the aging rate or aging speed over a long period of time, rather than detecting software aging symptoms or quantifying the severity of software aging at a given time instant.

This paper is intended to complement the inspection-based studies. First, we report aging phenomenon observed in our experiments. Second, Frequency Fourier Analysis (FSA) is employed to diagnose the aging symptoms at runtime. Finally, a metric for estimating severity of aging is extracted

based on the deviation from healthy running state, and an approach for online aging detection is proposed.

2. Related Studies

Software aging researches are aimed to counteract the effect of aging. It can be divided into four questions: What is the nature or root origin of software aging; how to construct a mathematical model to describe the evolution of software aging; how to measure software aging; how to control software aging. The relationship of those questions is depicted in Figure 1.

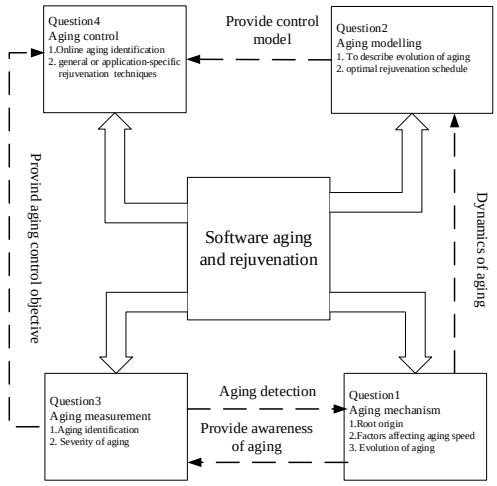

Figure 1. Questions of software aging.

From Figure 1 we can see that, aging mechanism study make us aware of the root origin of aging, the factors affecting aging speed, and the evolution of aging. Aging modelling tend to describe the evolution of aging by a mathematical model with some assumptions. Further, the mathematical model is solved to obtain optimal rejuvenation schedule. In addition, aging modelling can provide a precise control model for aging control studies. Moreover, aging control need quantified relationship between the overhead and benefit of rejuvenation. This can be answered by aging measurement studies. In addition, aging measurement studies can provision aging indicators which are meaningful for aging mechanism studies.

Model-based studies are the majority of software aging research. With the rich mathematical tools, model-based studies can provide useful hints for engineering practice. Nevertheless, the disadvantages of model-based studies are also obvious. Usually, model-based studies make some assumptions on the nature or root origin of software aging. These assumptions can hardly be validated by engineering practice. In fact, the aging process is much more complicated than what a mathematically tractable model may describe. In [4], a three-state stochastic model describe the aging process, with intention of obtain the optimal rejuvenation time. This model was extended and studied in detail by many researchers to answer similar question [6, 11].

Aforementioned questions 1, 3 and 4 in Figure 1 may fall into the category of inspection-based studies. These studies focus on practical software system, in which the data of interest are generated, collected and analyzed, with the purpose of forecasting possible incoming aging, understanding the mechanism of software aging, and/or quantitative evaluation of aging effect. The rationale behind inspection-based studies is that aging phenomenon is significantly related to available resources of computer system [3, 7, 10]. Shereshevsky *et al.* [9] monitors the Hölder exponent (a measure of the local rate of fractality) of the system parameters and find that system crashes are often preceded by the second abrupt increase in this measure. Vaidyanathan and Trivedi [10], a reward function is defined based on the rate of resource consumption, to estimate time to exhaustion for each resource. A metric “estimated time to exhaustion” is proposed to predict the approximate time of system resource depletion. A comprehensive evaluation function is proposed in [5] to measure the mean aging speed of the apache server. The proposed metric abstracts seven important resource usage parameters of system into two primary components via Principal Component Analysis (PCA) method. The two primary components are then combined to a hybrid metrics, namely the z-metric to represent the average aging speed along the runtime of the system.

Software metrics play a major role in estimating the quality of software [8]. The motivation of this paper lies in the fact that, metrics or measurements of software aging are rarely reported. Further, those metrics proposed in previous literatures only represent the mean aging speed over a relatively long time. Nevertheless, when the subject software becomes aged is not clearly stated. Moreover, how to detect aging and estimate its severity is not well studied. This paper is aimed to answer above questions based on observations in controlled experiments.

3. Datasets

The aging phenomenon discussed in this paper is shown in Figure 2, which is reported but not in-depth discussed in our previous study [5]. The data collection process in [5] can be described as follows: three workstations are used as clients to generate artificial requests to a web server (Apache httpd2.0). All four workstations are connected with a LAN. The system activity of the server is collected periodically. In order to expedite aging, we first test the capacity of web server under experimental environment. Then the three clients generate artificial requests slightly higher than the capacity to the web server. Finally, we collect many parameters by using a usage collection tools, which is running on the tested server. The objects or parameters collected on the workstations include those that describe the state of the operating system

resources, state of the processes running, information on the /tmp file system, availability and usage of network related resources, and information on terminal and disk I/O activity. Dozens of such parameters were collected at regular intervals (5 mins) for more than 400 hours.

Figure 2 shows the available physical memory (denoted by availablememory) of our subject software system under test. It should be noted that, our research focuses on the whole software system, i.e., including operating system and subject software. The available memory refers to that operating system can use as computing resources. The rationale behind this proposal lies in the fact that, the performance degradation refers to that of the whole computer system.

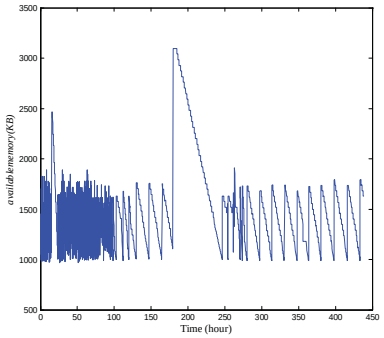

Figure 2. availablememory observed in our experiment.

From Figure 2 we can see that, the availablememory fluctuates with time from the very beginning, not showing obvious increasing or decreasing trend. This is because we apply too much requests on the web server, i.e., the web server is overloaded all the time. Usually, Apache httpd will spawn more child processes to handle the requests waiting in the request queue. Hence, too much physical memory is occupied by the child processes. In addition, if the workload of a server is too much, the network throughout will greatly increase consequently. Linux OS will allocate more physical memory as buffer or cache to improve the performance of web server.

In Figure 2, the vibratory curve tell us that the physical memory of computer system is nearly used up. Hence, Linux OS will use the swap space as memory to store data. The vibratory signal illustrate the page swapping mechanism between swap space and physical memory.

However, the vibration frequency in Figure 2 is decreasing gradually except for a period of abnormally increase at about 200th hour. More specifically, availablememory shows an obviously higher vibration frequency in the first 100 hour than that from 100th hour to about 280th hour. The vibration frequency becomes slightly even from 280th hour to the last.

As aforementioned, when the physical memory is nearly used up, Linux will swap out some out-of-date pages to disk, which will be swapped in while they are used again. Hence, the peaks in Figure 2 refers to the value of availablememory just as some pages are swapped out, and the valleys refer to that just as some pages are swapped in. Because a set of pages cannot be used or processed by CPU until they are swapped in, the swapping period, i.e., time from each peak to the next closest valley, can be employed as an indicator of performance. Thus, the obvious flattening trend of vibration frequency shown in Figure 2 drives us to extract a metric to represent the performance level of whole computer system.

4. Our Approach

4.1. FFT-Based Fault Diagnosis

Vibratory signals in engineering practice have been reported widely. These vibratory signals are usually obscure although it implicates much useful information. Frequency Fourier analysis can disclose the detailed information implicated in vibratory signals. In fact, frequency Fourier analysis has been widely employed to detect/diagnose the faults of machines [2].

For example, we can take the following steps to diagnose whether a machine is broken. First, a set of parameters of the machine are defined to describe the health of that machine. The vibratory signals of those parameters are collected when the machine is working in healthy state. Accordingly, the frequency spectrum of those signals should be calculated, which is used as "health indicator". Second, the frequency spectrum of those signals at specific time instant will be calculated, and will be compared against the health indicator. Finally, the deviation from the health indicator will be captured and used as a metric to indicate whether the machine is broken. For example, a nut wearing can introduce much low frequency signal into the vibratory signal of machine.

Inspired by the fault detection method used in mechanical system, we make a qualitatively analysis of Figure 2 based on Fourier analysis. First, we divide the curve in Figure 2 into ten parts. Each part represents the working state at that time. Then, we calculate the amplitude spectrum of each part of curve, and define the amplitude spectrum of the first part as healthy state. That is to say, we assume that the computer be not aged when it is restarted. Finally, we found the amplitude of low frequency part increases with each part of curve. That is an important sign of aging.

4.2. Amplitude Spectrum Based Aging Detection

In its narrowest sense, the Discrete-Time Fourier Transform (DTFT) of a time series is a decomposition of the series into a sum of sinusoidal components. The coefficients of which is the discrete Fourier transform

of the series. Given a discrete set of real or complex numbers: $x(n), n \in Z$, the DTFT of $x(n)$ is usually written as:

$$X(v) = N^{-1} \sum_{n=0}^{N=1} x(n) e^{-i 2\pi (v/N) n} \quad (1)$$

Where υ=0, 1, …, *N*-1.

If the sampling interval is T, the frequency measured in hertz is *v*/*N*T. The power spectrum refers to the square of amplitude of each sine or cosine signals, as accounts for the weight of various-frequency signals. A Fast Fourier Transform (FFT) is an efficient algorithm to compute the discrete Fourier transform. This algorithm requires the number of time series be power of 2. Adoption of the number of data employed in the FFT is a trade-off between robustness and sensitivity, i.e., the larger the number of data employed, the result is more immune from outliers and the result is more insensitive. In our case we use 512 data, which can eliminate the effect of outliers at about 200th hour shown in Figure 2. In our experiments, the sampling interval is 300 seconds, and the *N*=512, so the range of frequency is 0 to 0.00333 hertz.

The experiences of mechanical fault diagnosis tell us that, the distribution of amplitude spectrum in low frequency part, intermediate-frequency part and high frequency part, may implicate fault symptom. Because Fourier Transform is an even and periodical function, half frequency range, i.e., 0 to 0.00166 hertz can implicate most of information. Accordingly, the frequency range is divided into four sections: 0-frequency signal (0 hertz), low frequency signal (6.5×10-6 to 5.53×10-4), intermediate-frequency signals (5.53×10-4 to 1.10677×10-3) and high frequency signals (1.10677×10-3 to 1.6 to 10-3), in which 0-frequency signal represents the mean of original data in time domain.

In order to analyze the variation of amplitude spectrum with time, we equally divide our collected time series of availablememory into ten sections to make an off-line analysis. For example, section 1 corresponds to the data from very beginning to the 42th hour, section 3 corresponds to the data from the 85th hour to 127th hour. The last 139 data are discarded.

Then we transform the original signal in time domain to frequency domain to illustrate their amplitude spectrum distribution. Due to limited space of this paper, we only list the transformed results in sections 1, 2 and 10, which are corresponding to Figures 3, 4 and 5. In each figure, the upper subfigure represents availablememory signal in time domain. The bottom subfigure represents that in frequency domain. Take Figure 3 as an example, the upper subfigure shows the availablememory from the very beginning to 42th hour observed in our experiment. Corresponding to the upper subfigure in Figure 3, the bottom subfigure shows the amplitude distribution with respect to frequency.

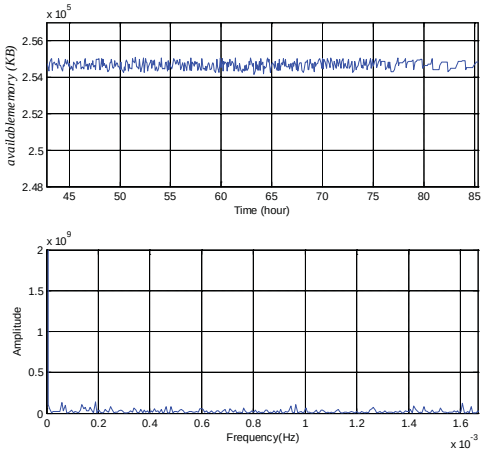

Figure 3. Amplitude spectrums in section 1.

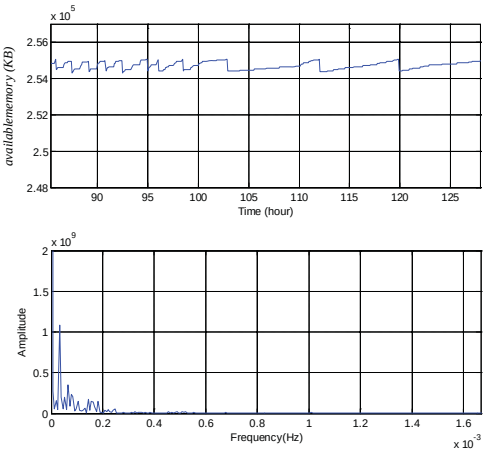

Figure 4. Amplitude spectrums in section 2.

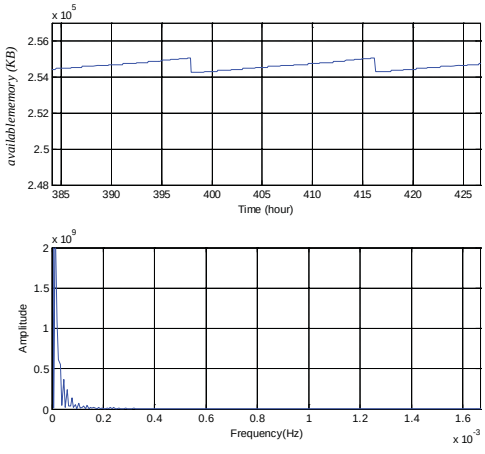

Figure 5. Amplitude spectrums in section 10.

Amplitude spectrum denotes the "amplitude distribution" with respect to frequency. In principle, the amplitude in low frequency part represents the "flattening degree" of a curve. More specifically, if the low-frequency amplitude is higher, the curve in time domain is flatter.

In comparison with the Figures 3 and 4 shows an increasing trend in its low frequency signals, i.e., the amplitude of low-frequency signal is increasing. This trend continues to grow in section four to section ten. Due to limited space of this paper, only section ten is listed, which is shown in Figure 5. From Figure 5, we

can see that, the amplitude of low frequency signals increases furthermore, and that of intermediate-frequency signals and high frequency signals decrease obviously. This trend illustrate that the computer will spend more time to swap a set of pages from swap space to physical memory and vice versa. This phenomenon can be interpreted as the decrease of activity level.

5. Online Aging Measurement

5.1 Aging Metric

We follow the fault diagnosis steps mentioned above to measure the effect of aging. In software aging studies, the initial state of computer system can be treated as healthy. This is to say, when the computer is started, there is no error conditions result from aging-related bugs. In this paper, the amplitude spectrum of section 1, which is shown in the bottom subfigure in Figure 3, can be defined as healthy state of our subject web server. From the subfigure in Figure 3, we can see the even distribution of amplitude spectrum with respect to frequency. Hence, the increasing trend of amplitude of low frequency signals mentioned above, can be used to represent how much the state of subject software deviates from healthy state.

According to Plancherel theorem, sum of power of time-domain signals is equal to that of frequency-domain signals, which is formulated as follows:

$$\sum_{n=0}^{N-1} |x_n|^2 = \frac{1}{N} \sum_{k=0}^{N-1} X_k^2 \qquad (2)$$

Above equation tells us that, if the signals in time domain fluctuate around a constant, the sum of power of it will be a constant. Considering there is no obvious trend with time shown in Figure. 2, we employ power spectrum to measure the aging symptom. In this subsection, we propose a simply yet efficient metric to measure the identified symptom. For purpose of aging symptom identification, we designate the proportion of Power of Low Frequency Signal (PLFS) over Power of All Frequency Signals (PAFS) as a metric of software aging, which can be written as:

$$PLFS \,/\, PAFS = \frac{\sum_{v \in low\ frequency} x(v)}{\sum_{v \in all\ frequency} x(v)} \qquad (3)$$

It should be noted that the 0-frequency signal, i.e., the first item in Equation 1 is excluded in the metric, because 0-frequency signal represents the mean of original data in time domain. We designate the sum of *PLFS* and all frequency signals as *PAFS*.

As expressed in Equation 2, and recall that the amplitude of our data is nearly constant, we can determine the *PAFS* is nearly a constant. Then, we can use only *PLFS* as our aging indicator.

It should be noted that, the metric of aging refers to the current aging extent in any time instant. The calculation of this metric need a set of data in time domain. In engineering practice, we need detect the aging symptom as possibly quick as. Thus, an online analysis approach should be proposed accordingly.

5.2. Slidingwindow Based Aging Detection

As mentioned above, based on the observed increasing trend of power of low-frequency signals, we can estimate the severity of software aging. For online aging detection purpose, we employ slidingwindow Fourier Transform to estimate the severity of software aging at real time [2].

Accordingly, we propose a slidingwindow monitoring approach to indicate the change of aging pattern online. It can be described as follows:

- *Step 1*. Define sampling frequency f, size of sliding window s and movement steps *n*;
- *Step 2*. Calculation of i*f/s;
- *Step 3*. Divide the signals into low frequency, intermediate-frequency and high-frequency sections, equipartition is a simple method.
- *Step 4*. Calculate *PLFS* in w_1;
- *Step 5*. Move the window n steps forward;
- *Step 6*. Calculate *PLFS* in w_i;

In which, sampling frequency f refers to the collection interval in time domain in experiments. w_i refers to i^{th} window. s refers to the size of window. In principle, larger w may make our algorithm more insensitive from outliers. That is to say, larger s can decrease false alarm rate. Rather, too large s can make the aging symptom obscure, and injure the aging detection performance of our algorithm. In summary, adoption of w is a trade-off between sensitivity and false alarm rate.

In principle, the smaller the symbol *n* is, the faster our approach can detect possible aging symptom. Nevertheless, too small *n* will result in too much computation workload. For an extreme example, when the *n* is set to 1, and interval of data collection is two minutes; if the computation of our approach cost more than two minutes, then, there are several new data will not be calculated by our approach. Consequently, too small value of *n* will delay aging detection. Also, too large value of *n* will still delay aging detection. Thus, the optimal value of *n* should be determined by computation time of our algorithm. That is to say, the optimal result should be that, when the *PLFS* of current windows is calculated, our data collection agent just collect *n* data. In summary, the value of *n* should be determined by the following expression:

$$n = \frac{computation\ time\ of\ our\ approach}{interval\ of\ data\ collection} \qquad (4)$$

The metric *PLFS* can be treated as a metric to identify aging. In our case, w is set to 256, and *n* is set to 1. Because the computation of our approach will cost no

more than interval of data collection (five minutes). Finally, the calculated *PLFS* in each sliding window is shown in Figure 6.

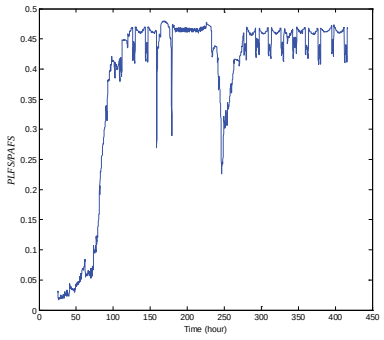

Figure 6. PLFS/PAFS of availablememory.

From Figure 6 we can see that, the resulting first point of our approach lies in about 21th hour. This corresponds to the fact that the size of window is 256 and interval of data collection is five minutes. From the very beginning to about 150th hour, *PLFS* increase sharply, showing an approximately monotonous property. Because the *PLFS* depicted in Figure 6 means swapping rate with time. Swapping rate describes how fast the pages are swapped from/to disk to/from physical memory. This is heavily correlated with performance of whole computer system. This metric can easily be applied for online aging diagnostic. By selecting different size of the sliding window, our approach provides a flexible trade-off between sensitivity of detection and robustness against outliers of the performance data. More specifically, the larger the size of window is employed, the calculated metric is more immune from outliers.

6. Conclusions

In this paper, we carry out controlled experiment to study software aging phenomenon in an overloaded web server. Based on the experimental observations, discrete Fourier analysis is employed to characterize the aging symptom. We found that the low frequency signal shows an increasing trend. This means the computer system cost more time to swap pages. This is an indicator of software aging, which is not reported by previous studies. Accordingly, a metric is proposed to evaluate the severity of aging. Moreover, we propose an approach for online aging measurement based on slidingwindow Fourier transform. The contribution of this paper can be summarized as follows:

- A metric is proposed to evaluate severity of aging.
- An online aging detection approach is proposed to detect possible aging.
- The methods in the field of signal processing is introduced to software aging study. This means the rich achievements in the field of signal processing can be employed to study the evolution of software aging.

Acknowledgments

This work is supported by the national key technology R and D program (Grant No. 2011BAH24B12) and the fundamental research funds for the central universities (Grant No. 3122013P006).

References

[1] Cai K., "Software Reliability Experimentation and Control," *Journal of Computer Science and Technology*, vol. 21, no. 5, pp. 697-707, 2006.

[2] Gertler J., *Fault Detection and Diagnosis in Engineering Systems*, Marcel Dekker, 1998.

[3] Grottke M., Li L., Vaidyanathan K., and Trivedi K., "Analysis of Software Aging in a Web Server," *IEEE Transaction on Reliability*, vol. 55, no. 3, pp. 411-420, 2006.

[4] Huang Y., Kintala C., Kolettis N., and Fulton N., "Software Rejuvenation: Analysis, Module and Applications," *in Proceeding of the 25th IEEE International Symposium on Fault-Tolerant Computing*, California, pp. 381-390, 1995

[5] Jia Y., Chen X., Zhao L., and Cai K., "On the Relationship Between Software Aging and Related Parameters," *in Proceeding of The 8th International Conference on Quality Software*, Oxford, pp. 241-246, 2008.

[6] Liang Y., Yang H., Fu J., Tan C., Liu A., and Zhu S., "The Effect of Real-valued Negative Selection Algorithm on Web Server Aging Detection," *Journal of Sofware*, vol. 7, no. 4, pp. 849-855, 2012.

[7] Matias R., Barbetta P., Trivedi K., and Filho P., "Accelerated Degradation Tests Applied to Software Aging Experiments," *IEEE Transaction on Reliability*, vol. 59, no. 1, pp. 102-114, 2009.

[8] Misra S. and Cafer F.,"Estimating Quality of JavaScript," *The International Arab Journal of Information Technology*, vol. 9, no. 6, pp. 535-543, 2012.

[9] Shereshevsky M., Crowell J., Cukic B., Gandikota V., and Liu Y., "Software Aging and Multifractality of Memory Resources," *in Proceeding of the 2003 International Conference on Dependable Systems and Networks*, San Francisco, pp.721-730, 2003.

[10] Vaidyanathan A. and Trivedi K., "A Comprehensive Model for Software Rejuvenation," *IEEE Transaction on Dependable and Secure Computing*, vol. 2, no. 2, pp. 124-137, 2005.

[11] Zhao J., Wang Y., Ning G., Trivedi K., Matias R., and Cai K., "A Comprehensive Approach to Optimal Software Rejuvenation," *Performance Evaluation*, vol. 70, no. 11, pp. 917-933, 2013.

Yun-Fei Jia is currently a lecturer in the School of Electronical and Information Engineering at Civil Aviation University of China. He received a B.E. (2001) and a M.S. (2004) degrees from HeBei University of Technology, and completed a PhD in software testing at BeiHang University in 2010. His research interests include software testing, software reliability modelling, cloud computing and software measurement.

Ren-Biao Wu was born in Wuhan, China, in 1966. He received a M.S. (1991) from Northwestern Polytechnical University, and completed phD in signal processing at Xidian University. Currently, he is a professor at Civil Aviation University of China. His interests are signal processing and image processing.