On the Effectiveness of Mann-Kendall Test for Detection of Software Aging

Fumio Machida
Knowledge Discovery Research Lab.
NEC Corporation, Japan
f-machida@ab.jp.nec.com

Artur Andrzejak
Institute of Computer Science
Heidelberg University, Germany
artur@uni-hd.de

Rivalino Matias, Elder Vicente
School of Computer Science
Federal University of Uberlandia, Brazil
rivalino@fc.ufu.br, elder@mestrado.ufu.br

***Abstract*—Software aging (i.e. progressive performance degradation of long-running software systems) is difficult to detect due to the long latency until it manifests during program execution. Fast and accurate detection of aging is important for eliminating the underlying defects already during software development and testing. Also in a deployment scenario, aging detection is needed to plan mitigation methods like software rejuvenation. The goal of this paper is to evaluate whether the Mann-Kendall test is an effective approach for detecting software aging from traces of computer system metrics. This technique tests for existence of monotonic trends in time series, and studies of software aging often consider existence of trends in certain metrics as indication of software aging. Through an experimental study we show that the Mann-Kendall test is highly vulnerable to creating false positives in context of aging detection. By increasing the amount of data considered in the test, the false positive rate can be reduced; however, time to detect aging increases considerably. Our findings indicate that aging detection using the Mann-Kendall test alone is in general unreliable, or may require long measurement times.**

***Keywords*—Software Aging, Trend detection, Mann-Kendall test**

I. Introduction

Software aging and rejuvenation research field aims to understand the phenomenology of software aging and develop effective countermeasures such as software rejuvenation. At the base of this process, a major requirement is the accurate detection of software aging effects. The variety and complex nature of software aging sources, make the correct detection of their effects a real challenge. Despite the importance of accurate aging detection, we observe a lack of preliminary studies evaluating the effectiveness of techniques used for this purpose.

Surveying the literature reveals that a common approach adopted in previous studies is based on the following general two steps. First, selected computer system metrics (e.g., free or used memory), which are assumed to capture aging effects, are monitored under controlled workload conditions. Next, based on the collected data set and supporting assumptions (e.g., aging is present if memory usage increase monotonically under constant workload), a statistical technique is used to detect trends that are considered as a sign of software aging. We highlight two aspects in applying this approach.

First, we observe that the monitored system variables are not always selected taking into account the specifics of the trend test used. This is a considerable risk, given that every trend analysis technique makes assumptions that need to be met; if assumptions are violated, test results can be erroneous. Specifically, in the presence of serial correlation, we know from the literature that many trend detection techniques can be estimated less accurately, which means the larger the correlation, the larger the uncertainty [1], [2]. Note that different system variables may demonstrate varying levels of autocorrelation in their time series. Hence, it is possible to conclude that not all system variables will provide time series with the necessary properties required by all trend tests, making both decisions strongly dependent of each other.

Second, the lack of theoretical or experimental validation of considered supporting assumptions. For example, several combinations of events not related to software aging may promote a monotonic increase of system memory usage under constant workload. In [3] an example of such situation in a real system is provided for this specific case. If the supporting aging assumptions are not verified, that is, grounded on a solid theory or consistent observations from previous experiments, the results can be misleading.

Based on the above findings from surveying the literature, in this paper we present the results of a study regarding the effectiveness of applying the Mann-Kendall trend test to software aging detection. We selected this test for our study given its broad citation in the software aging and rejuvenation literature (e.g., [4], [5], [6], [7], [8]). We also demonstrate how different system variables influence the quality of the test results.

The rest of this paper is structured as follows. Section II describes the problem context. Section III contains the description of our experimental plan, shows and discusses the results of our study. Finally, we state our conclusions in Section IV.

II. Mann-Kendall Test Applied to Software Aging Detection

A. Detecting aging via trend discovery

The Mann-Kendall test [9], [10] is a non-parametric test for detection of monotonic trends in time series data. In context of aging detection, such time series are traces of relevant computer system metrics (so-called *aging indicators*) such as Resident Set Size or Heap Usage (see Section III-A). As many types of aging (e.g., memory leaks) manifest via increased resource usage, positive (upward) slope of such indicators is commonly interpreted as a sign of software aging. Hence,

experimental studies of software aging have applied the Mann-Kendall test along with such indicators for the purpose of aging detection.

Our main criticism with respect to this approach is the observation that detected trend may not be caused by software aging. For example, if an aging indicator has large variance by nature, this test might produce multiple false alarms. Another cause for existence of a trend in a metric might be increased yet legitimate resource usage. This can have multiple causes, e.g. specific application design patterns or interaction of underlying system components, which may or may not be dependent on the workload.

The variance of the aging indicator values might be mitigated by increasing the amount of data (i.e., length of traces/time series) subjected to the test. To this aim, one can measure the system for a prolonged amount of time and apply tests repeatedly on all data since the beginning of measurements. We also experimentally evaluate this alternative approach against using Mann-Kendall test on a moving window of a maximal size. Results for both approaches are contrasted in Section III.

B. Mann-Kendall test

The Mann-Kendall test [9], [10] verifies the null hypothesis, H_0, indicating that there is no trend over time, against the alternative hypothesis, H_1, presenting there is an upward or a downward monotonic trend against zero slope. Let $Y = (t_i, y_i)$ be a time series data, where y_i is the series value at time point t_i (i is an integer index). Given *n* consecutive data points, the Mann-Kendall statistic S is computed by

$$S = \sum_{k=1}^{n-1} \sum_{l=k+1}^{n} sgn\left(y_l - y_k\right), \tag{1}$$

where sgn denotes the signum function

$$sgn\left(x\right) = \begin{cases} -1, & \text{if } x < 0 \\ 0, & \text{if } x = 0 \\ 1, & \text{if } x > 0. \end{cases} \tag{2}$$

The value of the S-statistic, computed via Equation (1), is compared against a critical value with respect to a given significance level α, where the critical values are drawn from standard tables (see e.g., [11]).

According to Kendall [10], a Normal approximation test could be used for data sets with more than ten values, providing that there are not many ties in Y (i.e., we have $y_l \neq y_k$ for any $l > k$). A standardized test statistic Z is used in this case and is compared to the Normal distribution.

C. Varying the amount of input data for S-statistic

While the Mann-Kendall test is robust against missing values, the amount n of data contributing to the S-statistic might be significant for reducing the impact of "noise" (e.g. variance) of aging indicators. We evaluate this aspect by contrasting the two approaches explained below. Their respective performance for trend detection is evaluated in Sections III-B and III-C.

1) Sliding window of limited size - MaxWinSize: The first approach, namely MaxWinSize, uses the limited number of consecutive values in time series data. In online monitoring of aging indicators the number of measurements increases over time. Here all the observed values are taken into account for the test until this number is less than or equal to $n_{\max}$. When the number of observations exceeds $n_{\max}$, the moving window is used to sample the most recent $n_{\max}$ values. Since the Mann-Kendall test is used with less than or equal to 40 samples in [11], we also implement this approach with $n_{\max} = 40$. Note that we can switch to the normal approximation test when we have enough sample points, but here we focus on the Mann-Kendall test using sliding window with limited number of samples.

2) Sliding window of unlimited size - UnlimWinSize: The alternative approach, namely UnlimWinSize, removes the size limitation in the first approach and uses all values from the initial data point (i.e., beginning of the measurement). This approach might be useful to detect the long time trend that is not clearly observed in a short time period. In addition, the accuracy of trend detection might be improved due to averaging out "noise" in the system metric. However, the test might become computationally more expensive than MaxWinSize approach, since now each data value needs to be compared to all subsequent data values.

III. Experimental Study and Evaluation

A. Experimental scenario

Our experiments focus on detecting software aging caused by memory leaks. For the controlled experiments, we create a synthetic workload generator (SWG) that emulates the behavior of a general-purpose application by requesting and releasing memory blocks, repeatedly, at random intervals (up to 30 seconds). Figure 1 presents the pseudo-code of SWG. The size of memory blocks is randomly sampled from a range of values according to the selected workload intensity; low, normal or high.

The SWG is programmed in C language, under Linux OS as the operating system, and uses the standard functions `malloc()` / `free()` for dynamic memory allocation. We injected a fault in free(), which leaks memory chunks in a given percentage, *p*, such that $p\%$ of `free()` calls fail to release the allocated memory. The leak rate, p, is also a controlled variable in SWG. If p is greater than 0%, the amount of memory consumption increases over time and it results in a memory leak.

Choosing a relevant aging indicator is a key to robust software aging detection. As discussed in previous works (e.g., [12], [13]), *Free Memory* (FM) is a system-wide aging indicator and hence its values might include some noises on actual aging trend. Application specific indicators such as *Resident Set Size* (RSS) and *Heap Usage* (HUS) are more precise to trace the actual memory demand from a target application. The values of FM and RSS are obtained by monitoring the Linux /proc directory, while the value of HUS is collected by instrumenting the memory allocator [14].

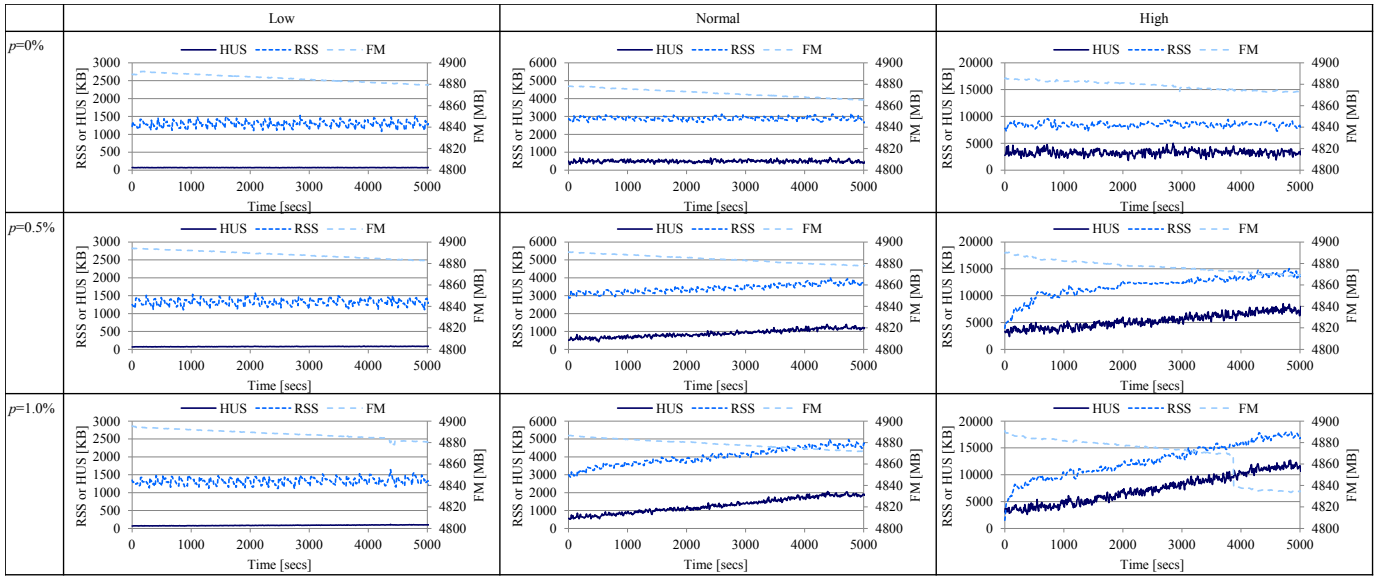

Figure 2. Observed time series data of the aging indicators Free Memory (FM), Resident Set Size (RSS), and Heap Usage (HUS) under workload generated by SWG for all combinations of leak rates ($p = 0\%$, 0.5%, or 1.0%) and workload intensity (low, normal, or high). The scale for RSS and HUS is on the left side of each chart, while the scale of FM is shown on the right side.

```
SWG (p, w, th)
  p: percentage of leak
  w: workload type
  st: thread status
  th: rate of thread creation
  rt: run time of application
  loop
    st = thread_create ( Load(p, w) );
    if (st != ZERO) then break;
    if (th == CONSTANT) then
       sleeps for 500000 microseconds;
    if (th == VARYING) then
       if (rt ≤ 30min.) then
          sleeps for 500000 microseconds;
       else if (rt ≤ 60min.) then
          sleeps for 250000 microseconds;
       else if (rt ≤ 90min.) then
          sleeps for 166666 microseconds;
          if (rt == 90min.) then rt = 0;

Function Load (p, w)
  t: sleep time in seconds
  k: leak activation factor
  c: allocated memory address
  t = random (1..30);
  if (w == LOW) then
     c = malloc (32 * random (1..16));
  if (w == NORMAL) then
     c = malloc (512 * random (1..55));
  if (w == HIGH) then
     c = malloc (1024 * random (1..200));
  if (c == NULL) then thread_exit;
  for each position in c
     c[position] = 0;
  sleeps for t seconds;
  k = random(1..100);
  if ( (p == 0.0) or
        (k ≤ (100 - p)) ) then
      free (c);
  thread_exit;
```

Figure 1. Implementation of the synthetic workload generator (SWG)

By changing the leak rate (0%, 0.5% or 1.0%) and workload intensity (low, normal or high), we obtained a set of time series data for FM, RSS, and HUS, shown in Figure 2. In each experiment, the values of aging indicators are collected about every nine seconds and we use the data observed until 5000 seconds after the start of the software execution. The FM values decrease over time regardless of the leak rate. On the other hand, the values of RSS and HUS gradually increase for leak cases (i.e. $p = 0.5\%$ or $p = 1.0\%$).

B. Results for MaxWinSize

First we evaluate the Mann-Kendall test with inputs from moving windows of maximum size 40 (i.e. MaxWinSize-approach, see Section II-C). The results are summarized in Figure 3. Plotted dot at a specific time point shows that the Mann-Kendall test has confirmed the trend existence (i.e. H_0 is rejected) with a confidence level of 95%. In every chart, results for each of the aging indicators (FM, RSS, and HUS) are plotted. All combinations of leak rates ($p = 0\%$, 0.5% and 1.0%) and workload intensities (low, normal, high) are shown.

For the aging indicator FM (at the topmost of the charts), trends are detected almost all the time, regardless of leak rates and workload intensities. As a result, we conclude that FM is considered to be hardly suitable for aging detection using the MaxWinSize-approach.

For the non-leak case ($p = 0\%$), trend detection is less frequent in the cases of RSS and HUS than the case of FM, although there are still some false alarms. The workload intensities also affect the amount of false alarms as higher workloads have more false alarms in both RSS and HUS. HUS produces less false alarms when compared with RSS, hence HUS is more robust to false alarms of memory-related aging.

For the leak cases ($p = 0.5\%$ and $p = 1.0\%$), the test should

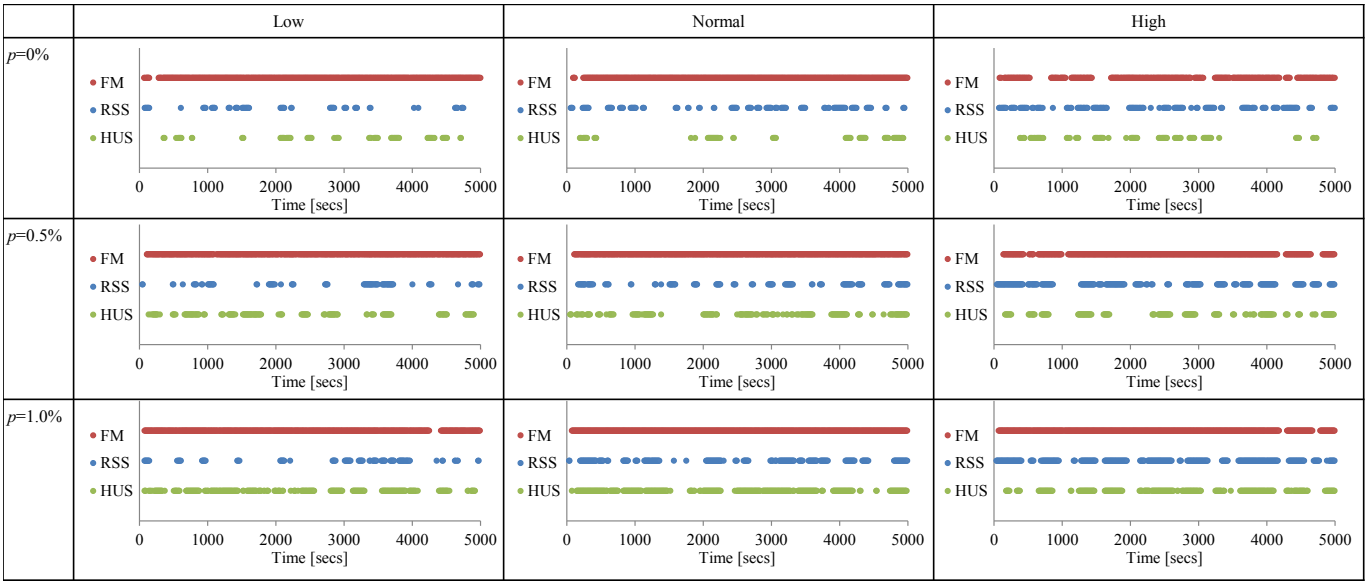

Figure 3. Time charts plotting the aging detection time points for MaxWinSize-approach; a plotted dot shows that the Mann-Kendall test has confirmed trend at this time point (plotted separately for FM, RSS, and HUS)

reject the null hypothesis as early as possible. However, both RSS and HUS cause a lot of false negatives (i.e., no trend detection), especially in case $p = 0.5\%$. The amount of false negatives is reduced for $p = 1.0\%$, but still it is quite likely that no leak is indicated, even after the measurement lasts for more than 3000 seconds. For low and normal workloads, HUS has lesser false negatives than RSS. The difference between HUS and RSS becomes blurred for high workload cases.

We observe that if the aging rate is constant, the trend detection performance of Mann-Kendall test does not improve with longer measurement duration. This can be attributed to the fact that in the MaxWinSize-approach the tests consider only a small fragment of recent values of the aging indicator, and the local profile of aging indicators does not evolve over time.

C. Results for UnlimWinSize

Next, we apply the Mann-Kendall test combined with UnlimWinSize-approach (i.e., test input is all available data since beginning of measurement, see Section II-C) to the same data set. The results are shown in Figure 4 (presentation conventions are the same as in Figure 3).

Also in this case FM is not useful to distinguish leak cases from the non-leak case, since the trend is detected almost all the time. Results for HUS in cases of low and normal workloads are satisfactory. The tests are reliable after 2000 seconds, as they can distinguish the leak cases from the non-leak case accurately. For high workload case, however, the test using HUS indicates many false alarms (i.e., for $p = 0\%$) after about 3000 seconds. Except for this specific scenario, the combination of HUS with the UnlimWinSize-approach achieves good performance of aging detection.

RSS as an aging indicator can also detect the trend with high confidence in normal and high workload cases. In high workload, however, it faces many false alarms in the non-leak case. Moreover, it almost completely fails to detect the trend in low workload case with leak rate $p = 0.5\%$.

D. Comparative analysis

To evaluate the performance of aging indicators for software aging detection, here we focus on the results of RSS and HUS, comparing the *sensitivity* and *specificity* of the experimental results. *Sensitivity* is defined as the fraction of true positives in cases of a true leaking. Figure 5 shows these sensitivities for the case $p = 1.0\%$. The UnlimWinSize-approach achieves higher sensitivity than the MaxWinSize-approach for all cases (i.e., different workloads combined with different aging indicators). The better result for the UnlimWinSize-approach can be explained by the increased input size of the data to be analyzed in the test.

On the other hand, *specificity* is defined as the fraction of true negatives in cases of no leaking. Figure 6 summarizes the results for both approaches (MaxWinSize vs. UnlimWinSize). Obviously there are no significant differences between the two approaches, only in selected cases MaxWinSize-approach is better (higher specificity).

However, if we take into consideration Figure 3 and Figure 4, we see that most of false positives appears only in an earlier phase of the experiments using the UnlimWinSize-approach (especially in low and normal workloads cases). We conclude that the UnlimWinSize-approach becomes more stable with longer runtime of measurements, i.e. its ratio of false positives might be reduced over time.

E. Lessons learned

As a brief summary of this section, we list the lessons learned from this study.

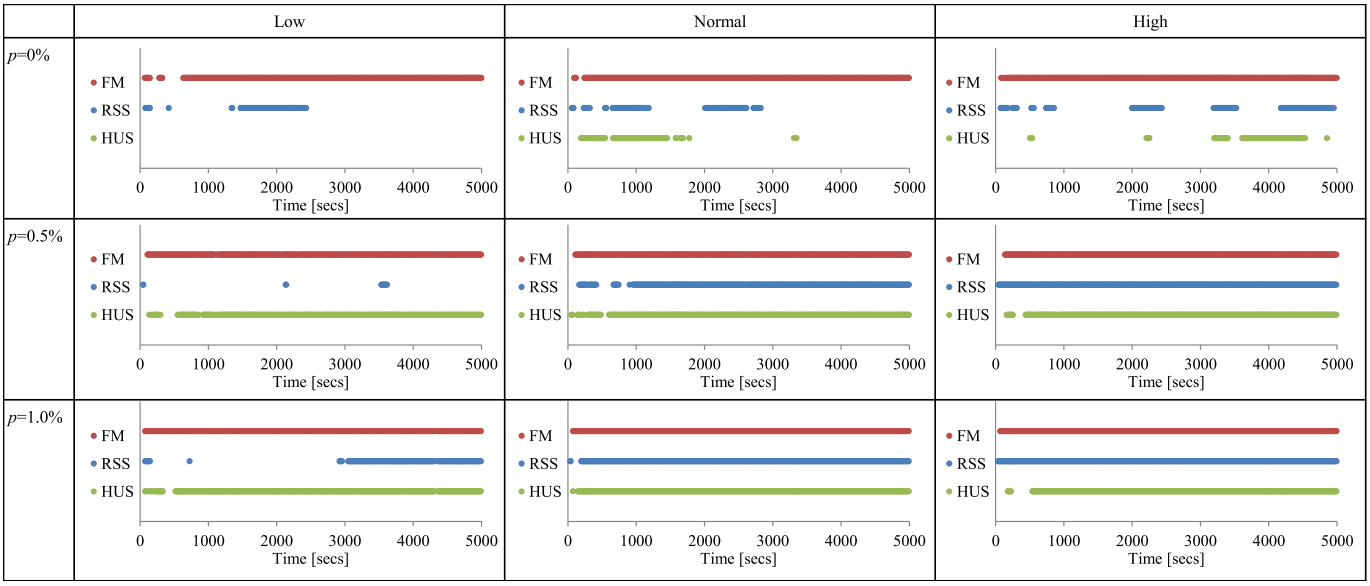

Figure 4. Time charts plotting the aging detection time point for UnlimWinSize-approach; a plotted dot shows that the Mann-Kendall test has confirmed trend at this time point (plotted separately for FM, RSS, and HUS)

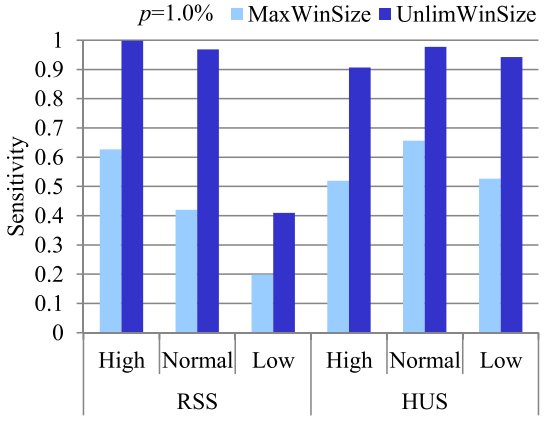

Figure 5. Comparison of MaxWinSize-approach vs. UnlimWinSize-approach by sensitivity (leak rate $p = 1.0\%$)

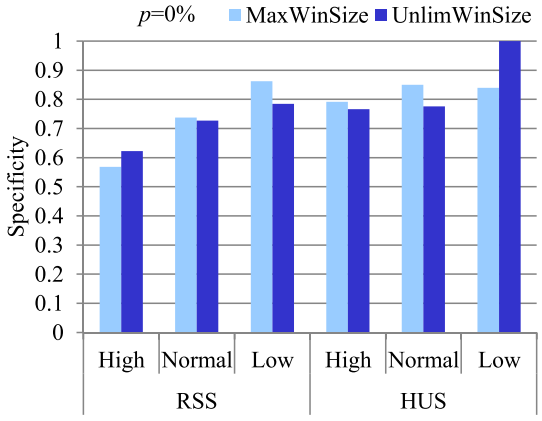

Figure 6. Specificity of MaxWinSize-approach vs. UnlimWinSize-approach (no aging, i.e. leak rate $p = 0\%$)

First, we compare the appropriateness of aging indicators. As observed in the results in the both approaches, the test using FM cannot distinguish the leak cases from the non-leak case. The detection performance is significantly improved by using RSS or HUS. The difference between these two metrics is not large. In detail, RSS has a slightly lower sensitivity and specificity, but this is also case-dependent.

The Mann-Kendall test with moving window for limiting the number of samples (MaxWinSize-approach) is not a reliable technique for aging detection without knowledge about the appropriate window size. In our experiments with 40 samples as a windows size, even with RSS or HUS used as aging indicator, the MaxWinSize-approach produces lot of false alarms in the non-leak case and false negatives in the leak cases. While the detection performance can be improved by increasing the window size, it is not a trivial issue to determine an appropriate window size. The primary reason is that the optimal size depends on various factors including the type of software aging, monitoring interval, aging indicator, and application workloads.

The Mann-Kendall test without any limitation of sample points (UnlimWinSize-approach) is more suitable for aging detection, but it needs more data to achieve high confidence and it depends on the quality of the used aging indicators. As the number of samples increases, the false alarms and false negatives are reduced. To distinguish the leak case from the non-leak case, sufficiently long period of observation is required.

Our experimental results clarify that applying trend tests alone is not enough to have accurate aging detection. Thus, in order to consider all details involved, we conclude that it is necessary to follow a comprehensive protocol for aging

detection. This protocol should provide clear procedures to answer questions such as:

- How to calculate the sample size with respect to the selected aging indicators and trend test technique?
- Which trend test is more appropriate for a given class of aging indicators?
- How to compute the risk of false-positives or false-negatives occurrences?

These are among others necessary questions to help the experimenter to conduct a more educated and reliable decision making.

IV. Conclusion

In this paper we studied the effectiveness of the Mann-Kendall tests applied to software aging detection as it has been used in previous works. We showed through an experimental study that the Mann-Kendall test suffers high rates of false positives, commonly indicating software aging even where there is no aging. This can be explained by the fact that variability (or "noise") of some underlying system metrics (aging indicators) creates short-term trends which are detected by the test.

To mitigate the latter effect, we contrasted the results of applying Mann-Kendall test to time series data from a moving window of maximum size 40 (MaxWinSize-approach) versus applying this test to all data since beginning of measurement until the current time point (UnlimWinSize-approach). Indeed, the UnlimWinSize-approach is more accurate and produces - in most cases - less false positives *after* a certain amount of metric data is available (in our case, after about 60% of the complete trace data). Thus, there is a trade-off of the UnlimWinSize-approach between running time and accuracy. This limits its utility for detecting aging during standardized software testing as suggested in [15], [16].

Our study has also uncovered that the choice of underlying system metrics as aging indicators has significant impacts on the aging detection capability. For example, the *Free Memory* metric turned to be useless (in our experimental setting) as it exhibits strong trends (see Figure 2, top row) and so always indicates presence of aging. On the other hand, the metrics *Resident Set Size* and *Heap Usage* turned out to be more useful.

Our future work will refine these results in several directions. We will broaden the study including other system metrics and experimental environment. Further explorations will consider the confidence in trend presence, not only a binary trend confirmation. Finally, we will further study the efficiency of aging detection by version comparison method [15].

V. Acknowledgments

This work is supported in part by the grant AN 405/2-1 entitled *Automated, minimal-invasive Identification and Elimination of Defects in Complex Software Systems* financed by the Deutsche Forschungsgemeinschaft (DFG), and Brazilian research agencies CNPq, CAPES, and FAPEMIG.

References

[1] S. Yue, P. Pilon, B. Phinney, and G. Cavadias, "The influence of autocorrelation on the ability to detect trend in hydrological series," *Hydrol. Process.*, vol. 16, no. 9, pp. 1807–1829, 2002.

[2] S. Yue and C. Wang, "The Mann-Kendall Test Modified by Effective Sample Size to Detect Trend in Serially Correlated Hydrological Series," *Water Resources Management*, vol. 18, pp. 201–218, 2004.

[3] R. Matias and P. J. F. Filho, "An experimental study on software aging and rejuvenation in web servers," in *Proceedings of the 30th Annual International Computer Software and Applications Conference - Volume 01*, COMPSAC '06, (Washington, DC, USA), pp. 189–196, IEEE Computer Society, 2006.

[4] S. Garg, A. Van Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," in *Proceedings of the The Ninth International Symposium on Software Reliability Engineering*, ISSRE '98, (Washington, DC, USA), pp. 283–, IEEE Computer Society, 1998.

[5] G. Carrozza, D. Cotroneo, R. Natella, A. Pecchia, and S. Russo, "Memory leak analysis of mission-critical middleware," *J. Syst. Softw.*, vol. 83, pp. 1556–1567, Sept. 2010.

[6] G. Tian, J. Zhan, C. Ren, and D. Meng, "A case study for soa system rejuvenation," in *PDPTA* (H. R. Arabnia, S. C. Chiu, G. A. Gravvanis, M. Ito, K. Joe, H. Nishikawa, and A. M. G. Solo, eds.), pp. 97–103, CSREA Press, 2010.

[7] F. Machida, J. Xiang, K. Tadano, and Y. Maeno, "Software life-extension: A new countermeasure to software aging.," in *ISSRE*, pp. 131–140, 2012.

[8] D. Controneo, R. Natella, R. Pietrantuono, and S. Russo, "A survey on software aging and rejuvenation studies," *IACM Journal on Emerging Technologies in Computing Systems (JETC), to appear*.

[9] H. Mann, "Nonparametric tests against trend," *Econometrica*, vol. 13, pp. 245–259, 1945.

[10] M. G. Kendall, *Rank Correlation Methods*. London: Griffin, 4. ed., 1970.

[11] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. John Wiley & Sons, 2 ed., 2001.

[12] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, pp. 1–6, IEEE, 2008.

[13] R. Matias, B. E. Costa, and A. Macedo, "Monitoring memory-related software aging: An exploratory study," in *ISSRE Workshops*, pp. 247–252, 2012.

[14] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal, "Dynamic instrumentation of production systems," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '04, pp. 15–28, 2004.

[15] F. Langner and A. Andrzejak, "Detecting software aging in a cloud computing framework by comparing development versions," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, (Ghent, Belgium), pp. 896–899, May 27-31 2013.

[16] F. Langner and A. Andrzejak, "Detection and root cause analysis of memory-related software aging defects by automated tests," in *21st IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2013)*, (San Francisco, California), August 14–16 2013.