Delft University of Technology
Software Engineering Research Group
Technical Report Series

Locating Performance Improvement Opportunities in an Industrial Software-as-a-Service Application

Cor-Paul Bezemer, Andy Zaidman, Ad van der Hoeven,
Andre van de Graaf, Maarten Wiertz, Remko Weijers

Report TUD-SERG-2012-013

TU Delft

SERG

TUD-SERG-2012-013

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Locating Performance Improvement Opportunities in an Industrial Software-as-a-Service Application

Cor-Paul Bezemer*, Andy Zaidman*, Ad van der Hoeven†, André van de Graaf†, Maarten Wiertz†, Remko Weijers†
*Delft, University of Technology
{c.bezemer, a.e.zaidman}@tudelft.nl
†Exact
{ad.van.der.hoeven, andre.van.de.graaf, maarten.wiertz, remko.weijers}@exact.com

***Abstract*—The goal of performance maintenance is to improve the performance of a software system after delivery. As the performance of a system is often characterized by unexpected combinations of metric values, manual analysis of performance is hard in complex systems. In this paper, we extend our previous work on performance anomaly detection with a technique that helps performance experts locate spots — so-called performance improvement opportunities (PIOs) —, for possible performance improvements. PIOs give performance experts a starting point for performance improvements, e.g., by pinpointing the bottleneck component. The technique uses a combination of association rules and several visualizations, such as heat maps, which were implemented in an open source tool called WEDJAT.**

In this paper, we evaluate our technique and WEDJAT in a field user study with three performance experts from industry using data from a large-scale industrial application. From our field study we conclude that our technique is useful for speeding up the performance maintenance process and that heat maps are a valuable way of visualizing performance data.

I. INTRODUCTION

In the ISO standard for software maintenance[1], four categories of maintenance are defined: corrective, adaptive, perfective and preventive maintenance. Perfective maintenance is done with the goal of improving and therefore perfecting a software system after delivery. An interesting application of perfective maintenance is *performance maintenance*, as many software performance issues become obvious after deployment only. While a large amount of research has been done on software performance engineering in general [1], only few papers deal with software performance maintenance. In addition, experience from industry shows that performance engineers mainly use combinations of simple and rather inadequate tools and techniques rather than integrated approaches [2], making performance maintenance a tedious task.

Perfecting software performance is typically done by investigating the values of two types of metrics [2]. On one hand, high-level metrics such as response time and throughput [3] are important for getting a general idea of the performance state of a system. On the other hand, information retrieved from lower-level metrics, e.g., metrics for memory and processor usage — so called performance counters [4] —, is important for pinpointing the right place to perform a performance improvement. However, determining a starting point for analysis of these lower-level metrics is difficult, as the performance of a system is often characterized by unexpected combinations of performance counter values, rather than following simple rules of thumb [5]. This makes manual analysis of performance in large, complex and possibly distributed systems hard.

In previous work, we have presented an approach for detecting performance anomalies using performance counter measurements [6]. This approach allows us to detect performance anomalies by identifying system states at which the system is performing relatively slow based on performance counter values. We have also shown that our approach allows faster detection of problems than a 'traditional' threshold setting for the average response time. In this paper, we extend this approach with a technique that helps performance experts locate spots for possible performance improvements. Our technique describes such spots as *performance improvement opportunities* (PIOs). PIOs give performance experts a starting point for performance improvements, e.g., by pinpointing the bottleneck component. The technique uses a number of visualization methods, amongst which heat maps [7], to provide a compact overview of the performance history of a system. We have implemented this technique in an open source tool called WEDJAT, which we present and evaluate by conducting a field study with performance experts from industry in this paper. Our technique is a high-level approach: it works complementary to lower level approaches such as profiling [8], as it helps narrow down the server or hardware which requires more investigation.

This paper is organized as follows. In Section II we discuss relevant background information. In Section III we present our idea of using heat maps for performance analysis. The implementation of this idea is presented in Section IV. We evaluate our approach using a field user study (Section V) and present the results in Section VI. Results are discussed in Section VII. We present related work in Section VIII and we conclude our work in Section IX.

II. BACKGROUND

In previous work [6] we presented an approach that allows to identify performance anomalies, e.g., sudden slowdowns, in a software system using low-level performance measurements only. That work contrasts earlier studies that used the response time as the main indicator of performance anomalies. We have shown that by using low-level performance measurements, we were able to efficiently identify performance anomalies that

[1]http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39064

originate at the server. Furthermore, our technique worked very precisely in that it keeps track of performance profiles per user, so that a user with many database records versus a user with a relatively low number of database records gets treated differently. While we could very precisely identify when a performance anomaly occurred, we had no means of identifying its cause and that is what this paper adds.

In the next two subsections we briefly describe our approach from [6]. As our approach is a learning-based approach, we first describe the training phase in Section II-A, before we go over to the actual classification phase in Section II-B.

A. Training Phase

Central to our approach [6] are the `SAratio` and the `intensity` metrics. The `SAratio` (or *Slow-to-All-actions-ratio*) for a time interval t is defined as:

$$SAratio_t = \frac{|SLOW_t|}{|SLOW_t| + |NORMAL_t|}$$

We define an action as slow when it belongs to the 15% slowest actions in terms of response time for a particular user of a particular application (or feature) for a time interval t. We now calculate the `SAratio` for all time intervals of the training period using a sliding window approach. As we now have a `SAratio`-value for all monitored time intervals, we can identify intervals during which the system was running relatively slow.

The next step is to define thresholds for the `SAratio`, such that we can classify system load for each interval as:

- $high$: system load is typically too high, which makes it perform slow (highest 5% of values for `SAratio`)
- med: system load may become or may just have been problematic (medium 10% of values for `SAratio`)
- low: system load is non-problematic (low 85% of values for `SAratio`)

After classifying all intervals as exhibiting $high$, med, or low load based on the `SAratio`, we assign the performance counter data to the time interval during which it was monitored. Next, we use association rule mining to classify the state of the system using performance counters only (e.g., memory$\geq$80, CPU$>$70 $\rightarrow$ $high$). In particular, we want to use the low level performance counter measurements rather than the $SAratio$ directly as they can give a more precise description of the performance, which can assist in giving a diagnosis.

B. Classification Phase

During the classification phase we classify new performance counter measurements. However, because we are not interested in isolated spikes in the performance of a system, but rather in situations in which the system is relatively slow for longer periods of time, we use a sliding-window approach to filter out these isolated spikes.

For a sliding window of size n, we determine which association rules match for the measured performance counter values. Next, we count how many of those roles are classified as low, med and $high$ load (see Section II-A). Finally, we determine the $intensity$ metric as follows:

$$\text{If } >30\% \text{ of the classifications in the window } \begin{cases} \text{high} & \rightarrow & \text{intensity+2} \\ \text{med} & \rightarrow & \text{intensity-1} \\ \text{low} & \rightarrow & \text{intensity-2} \end{cases}$$

This results in a series of values for the intensity of the load on a system during the classification phase, in which new performance counter measurements are classified. Ideally, the value of this metric is zero; whenever the value is larger, we have an indication that the system is running relatively slow.

Both the association rules used to calculate the intensity metric and the intensity metric itself will form the basis for the approach presented in this paper.

III. Approach

In this section, we present our approach for locating performance improvement opportunities (PIOs), which is an extension of our approach for performance anomaly detection explained in the previous section. A PIO is a snapshot of the system during a period of time at which the performance of the system could possibly be improved. This snapshot is described by the following for that period of time:

- Date and time of start of the PIO
- Date and time of end of the PIO
- Intensity graph (Section II-B and Figure 2)
- Raw metric value matrix (Section III-A)
- Rule coverage matrix (Section III-A)
- Missing value matrix (Section IV-E)

A PIO description can assist performance engineers in performing perfective maintenance by pinpointing the bottleneck component during the PIO. The next step could be investigation of that component using a profiler (see Section VIII).

Our PIO detection approach exploits the association rules used during the classification process of the anomaly detection to detect starting points for exploring possible performance improvement opportunities. The goal of our approach is to analyze the information in the rules matched by a measurement and detect clusters of performance counter metrics that help us to decide on which server or hardware we must start looking for possible performance improvements.

Table I shows a sample set of association rules used to characterize the load on a system into the classes low, med and $high$. The system consists of server $S1$ with performance counters $PC1$ and $PC2$ and server $S2$ with performance counter $PC1$. Table I contains a set of sample measurements for these performance counters as well. As defined in [6], a $high$ load often represents a performance anomaly, which indicates a possible PIO. We exploit this property to get an indication of the bottleneck component.

TABLE I
Sample association rule set and performance counter measurements

Sample association rule set	**Sample measurements**			
1 S1PC1$>$80 & S2PC1$<$60 $\rightarrow$ $high$	t	S1PC1	S1PC2	S2PC1
2 S1PC1$>$70 & S1PC2$>$70 $\rightarrow$ $high$	0	40	60	80
3 S1PC1$>$90 $\rightarrow$ $high$	1	95	60	80
4 S1PC2$<$30 $\rightarrow$ med	2	98	80	80
5 else $\rightarrow$ low	3	98	95	55
	4	98	80	80
	5	40	25	80

A. The Rule Coverage Matrix

Our approach uses a matrix m with one row for each performance counter and one column for every time t we receive a measurement. This matrix contains the raw values monitored for each counter.

In addition, we maintain a so-called coverage matrix m'. The rows of this matrix contain the performance counters, the columns depict measurements. The first column, representing the first measurement is initialized to 0. Each time a new measurement is received, the last column of m' is copied and the following algorithm is applied:

- Increase $m'_{i,j}$ if performance counter i is covered by a $high$ rule at measurement j.
- Leave $m'_{i,j}$ equal to $m'_{i,j-1}$ for a med rule
- Decrease $m'_{i,j}$ if performance counter i is covered by a low rule at measurement j, with a minimum value of 0

Note that the original 'raw' values of the performance counters in m are left untouched in this process. We update the value of every $m'_{i,j}$ only once for every measurement, even though multiple covering rules may contain the same performance counter.

The rationale behind building the rule coverage matrix this way is the following:

1) The ruleset describes all known cases of when the system was performing slowly.
2) We expect all measurements made during a PIO to be covered by the same, or similar rules when they are classified. The reason for this is that performance counter values are in general relatively stable, which means that abnormal values of (combinations of) performance counters will be exhibited for a longer period of time, i.e., throughout the PIO.
3) When entering this into the rule coverage matrix this way, higher values in m' will appear because these values will be increased for performance counters which occur in adjacent measurements.
4) Eventually, clusters of higher values in m' for performance counters on specific hardware will appear.
5) These clusters can be used to do performance maintenance, e.g., by pinpointing a bottleneck component.

The following example illustrates this. Figure 1(a) shows the resulting m' after applying our approach to the measurements and ruleset of Table I. Figure 1(b) shows a visual representation of this matrix in the form of a heat map [7]. In such a map darker colours represent higher values. In our simple example we can see a cluster of dark coloured performance counters at server $S1$, indicating this server may be a bottleneck.

In the next section we present WEDJAT, a tool which implements the rule coverage matrix and heatmap. In addition, in WEDJAT we combine a number of methods for visualizing performance counter data.

IV. TOOL IMPLEMENTATION: WEDJAT

In our approach, the starting point for all investigations is the intensity metric discussed in Section II. Whenever the intensity is larger than 0, the performance data requires more detailed inspection. Our tool WEDJAT[2] gives an overview of performance data deviating from its normal behaviour, helping performance experts to quickly identify possible bottlenecks.

pc \ t	0	1	2	3	4	5
S1PC1	0	1	2	3	4	4
S1PC2	0	0	1	2	3	3
S2PC1	0	0	0	1	0	0
covered by rules #	5	3	2,3	1,2,3	2,3	4

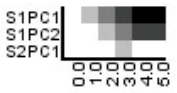

Fig. 1. Rule coverage matrix for Table I and the corresponding heatmap

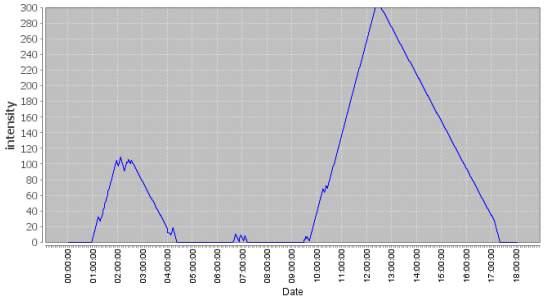

Fig. 2. Intensity graph showing two performance incidents

In WEDJAT, we combine our ideas of using a rule coverage matrix and heat maps to visualize performance data. WEDJAT offers several views, which can be used on their own or complementary to each other. In the remainder of this section, the components of WEDJAT and its views are presented.

A. Time Slider

One of the main components in WEDJAT is the time slider, which enables a time interval for the heat maps to be selected. This allows a user to zoom in or out on interesting events.

B. Intensity Graph View

Goal The intensity graph is intended to be displayed in a performance monitor or dashboard and its purpose is to serve as a trigger for starting investigation with WEDJAT. We have included the intensity graph in WEDJAT itself as well; the time slider allows to zoom in or out on this graph.

Interpretation Whenever this graph shows a peak (value greater than zero), a possible PIO exists and the performance data requires deeper investigation.

Example Figure 2 shows the intensity graph generated during two performance incidents in a system.

C. Rule Coverage Heat Map View

Goal The rule coverage heat map is the visualization of the rule coverage matrix as described in Section III. The goal of this heat map is to give an indication of where to start the investigation for possible performance improvements.

Interpretation In this heat map, adjacent dark squares indicate that this counter occurred in matched association rules for a longer period of time.

[2]The Wedjat, or Eye of Horus, is an ancient Egyptian symbol of protection and good health.

Example Figure 3 depicts a rule coverage heat map in which the rule (`sv4/Memory/Available MBytes/null` $\geq$ 691) `and` (`ws4/Processor/% Processor Time/_Total` $\leq$ 53.519428)[3] was matched repeatedly.

D. Deviation-from-mean Heat Map View

Goal The goal of this heat map is to easily identify groups of abnormal values for performance counters.

Interpretation In the deviation-from-mean heat map, the intensity of the color of its squares is calculated based upon the number of standard deviations from the mean value of the performance counter represented by that row. In this heat map, adjacent dark squares indicate that this counter exhibited an unusually low or high value for a longer period of time. WEDJAT offers two possibilities of filtering deviation-from-mean heat maps: show only the counters in the rule coverage heat map and show all counters for a specific server. It is possible to display the heat map for two servers at the same time, for example, to compare them to verify a load balancer is behaving properly.

Example The deviation-from-mean heat map for server `ws4` is displayed in Figure 4. In this heat map it is clear to see that there was a period of approximately 20 minutes during which the monitored processor time counter exhibited abnormal behaviour (see first line between approximately 10:20 and 10:40).

E. Missing Values Heat Map View

Goal During the classification phase, we classify a set of performance counter values. In some cases, this set is not complete, i.e., it does not contain values for all performance counters expected based on the configuration of the monitor. The reason for this may be a configuration error or a problem with hardware or a server. The goal of the missing values heat map is to detect such configuration and hardware issues.

Interpretation A value *MissingIntervals* is maintained for every performance counter, which is initialized to 0 and increased when we do not receive a value for this performance counter in a measurement interval. After a value is received *MissingIntervals* is reset to 0. In the missing values heat map, the intensity of the color of a square is based on the value of *MissingIntervals*. As such, adjacent dark squares indicate that this counter did not exhibit a value for a longer period of time, which indicates either a configuration or hardware issue.

Example Figure 5 shows the missing values heat map when there was a problem with the `sv3` and `sv2` servers and a configuration error on the SQL cluster.

F. Line Chart View

Goal After clicking on a square in the deviation-from-mean heat map, a line chart is plotted displaying the values of the performance counter in the selected time interval. In addition, the mean calculated during the training phase is shown. The goal of displaying the line chart is to allow users of the tool to combine the information in the heat maps with views they are more accustomed to.

Interpretation This chart can be used to see trends, e.g., whether the counter is increasing or decreasing rapidly.

Example Fig. 6 shows a line chart for counter `dbclus1/LogicalDisk/Avg. Disk sec/Read/W:`.

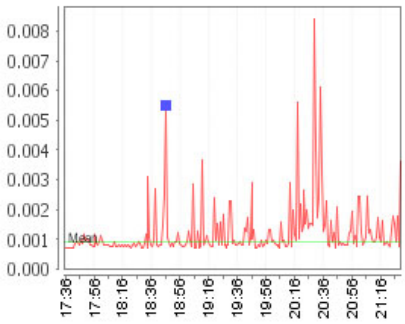

Fig. 6. Line chart example

G. Histogram View

Goal After clicking on a square in the deviation-from-mean heat map, two histograms are plotted as well. The goal of the histograms is to give the user a quick overview of normal values for the selected performance counter.

Interpretation The first histogram shows the values of the performance counter observed during the training phase. This histogram shows the user a quick overview of normal values for this performance counter. The second one shows the histogram of the values observed during the selected time interval. This histogram allows the user to compare the distribution of the values of this performance counter with those observed during the training period.

Example Figure 7 depicts these two histograms for the `dbclus/LogicalDisk/Avg. Disk sec/Read/U:` performance counter. The dark line in the histograms depicts the value of the selected square. From these histograms becomes clear[4] that during the trainingsperiod values between 0.025 and 0.005 rarely occured, while these occured frequently during the selected period. This could indicate that the counter is exhibiting abnormal behaviour during the selected period.

V. DESIGN OF THE FIELD USER STUDY

In this section, we evaluate WEDJAT and our idea of using heat maps for software performance maintenance in a field user study. In our study, we are looking for an answer to the following research questions:

RQ 1. *Does the rule coverage matrix provide a good starting point for performance maintenance?*

RQ 2. *Are heat maps an appropriate way of visualizing performance data?*

The outline of this section is as follows. First, we will describe the field setting of the user study. After this, we will discuss the setup of the study and the profile of the participants. In the next sections, we will present and discuss the findings of the field user study.

[3] In this paper, we use the format *ServerName/CounterCategory/PerformanceCounter/Instance* to describe a performance counter.

[4] Note that the scale of the axes of the histograms is different.

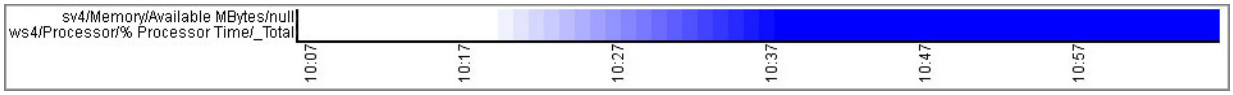

Fig. 3. Rule coverage heat map (rule matched repeatedly: (`sv4/Memory/Available MBytes/null` $\geq$ 691) and (`ws4/Processor/% Processor Time/_Total` $\leq$ 53.519428)

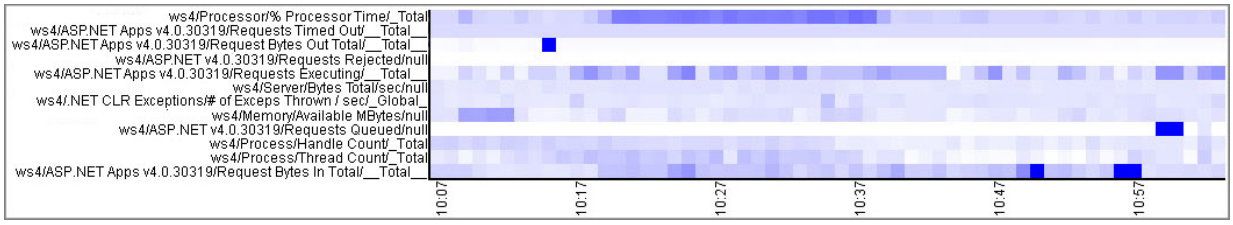

Fig. 4. Deviation-from-mean heat map for the server `ws4`

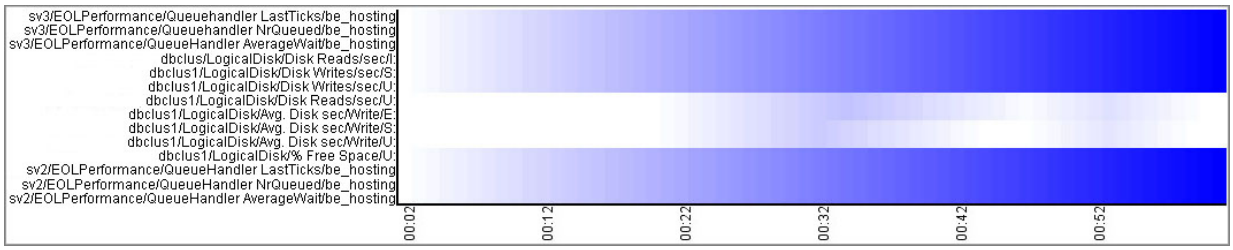

Fig. 5. Missing values heat map

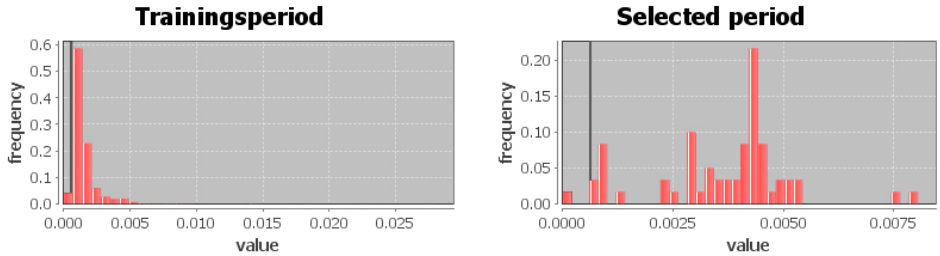

Fig. 7. Histograms for the `dbclus/LogicalDisk/Avg. Disk sec/Read/U`: performance counter

A. Field Setting

In order to find out how useful the rule coverage matrix and heat maps are for indicating starting points for improving performance bottlenecks, we set up a field user study with three performance experts from Exact[5]. Exact is a Dutch-based software company, which specializes in enterprise resource planning (ERP), customer relationship management (CRM) and financial administration software. Exact has over 1900 employees working in more than 20 countries. Founded in 1984, Exact has over 25 years of experience in multi-user client/server software and web applications. Since several years, Exact has also been offering a successful multi-tenant [9] Software-as-a-Service solution, called Exact Online[6] (EOL), which is the target of our field study. Exact Online uses ASP.NET and SQL Server.

B. Participant profile

All participants are performance experts in the EOL team. The first part of the questionnaire consisted of a set of general questions about their experience, knowledge and the current process at EOL. Their experience is summarized in Table II. Participant I and II have 16+ years of experience with software engineering and 10 years of experience with performance analysis. Participant III does not have direct experience as a software engineer, but does have 10 years of performance analysis experience from a product management point of view. All participants indicate their knowledge of the EOL infrastructure and of performance analysis in general as excellent.

TABLE II
YEARS EXPERIENCE OF THE PARTICIPANTS

Participant	Role	Soft. Eng.	Perf. Analysis	EOL
PI	Senior Research Engineer	16	10	7
PII	Principal Research Engineer	18	10	3
PIII	Product Manager	0	10	7

In the current situation at EOL, performance problems are noticed from log reports or follow from customer complaints about application speed and availability. Analysis of these problems and other possible opportunities to improve performance is done using a set of standard non-integrated tools

[5]http://www.exact.com
[6]http://www.exactonline.nl

such as Microsoft's PerfMon and SQL Profiler. All participants spend 5–48 hours per week improving the performance of Exact Online, depending upon the number of problems reported.

C. Experimental Setup

In a short preliminary study an initial evaluation of the usability of WEDJAT and its enclosed heat maps was done. In two sessions, three participants were requested to investigate the performance issue discussed in [6] using WEDJAT only. A large amount of feedback and improvements to the usability of the heat maps and WEDJAT were gathered from the participants. These improvements were implemented and evaluated during the field study. In the field study, three participants were requested to investigate a performance issue different from the one investigated during the preliminary study. Two of the participants to the field study took part in the preliminary study, while the third had never worked with WEDJAT before.

The performance data used during the field study was monitored during a period of one month on part of the Exact Online infrastructure. The association rule set used by WEDJAT was trained on the data monitored during the first week of that period. The monitored part of the infrastructure consists of 6 webservers, 3 service machines, 2 SQL clusters and 2 virtual domain controllers, exhibiting a total of 140 performance counter values every minute[7].

The process followed during the field study is discussed in the remainder of this section.

1) Step 1: Demo of WEDJAT*:* All sessions were started with a short demo of WEDJAT, demonstrating all its features.

2) Step 2: Assigned Task: The participants were asked to investigate a real performance issue in WEDJAT for around 1.5 hours. We gave technical assistance during the investigation and told them they would have to fill in a questionnaire at the end of the session about the usability of WEDJAT.

The task the experts are asked to solve is the following:

1) Load the data set for April 27 2012.
2) List the times of PIOs found on that date.
3) Investigate the PIO found between 10 AM and 11 AM.
4) Indicate the bottleneck component(s) found in that PIO.
5) Elaborate on how you selected the bottleneck(s).

The bottleneck component in the PIO between 10 AM and 11 AM was formed by the processor of a web server. This was expressed in WEDJAT by the following:

1) The `Processor/% Processor Time` performance counter of `ws4` (web server 4) showed a cluster of dark squares in the rule coverage heat map (see Figure 3).
2) The `Processor/% Processor Time` performance counter of `ws4` (web server 4) showed a cluster of dark squares in the deviation-from-mean heat map (see Figure 4).

3) Step 3: Questionnaire: After the investigation, the participants were asked to fill in a questionnaire consisting of 5 open questions and 56 questions that could be answered on a 5-point Likert-scale, ranging from 1 for 'strongly agree' to 5 for 'strongly disagree'. During the questionnaire, all participants regularly switched back to WEDJAT to answer the questions as accurately as possible. The questionnaire and the participants' answers are listed in Table III.

4) Step 4: Contextual Interview: A large amount of information and feedback was elicited from the participants through the use of a contextual interview [10]. During such an interview, the study participants are requested to solve an assigned task while they are being questioned constantly when working with the application for suggestions and feedback based on their actions. This allows them to articulate their normal or preferred work practices [10] and to get involved in the design of the application. During the session with two participants, we encouraged them to cooperate and discuss their ideas out loud to elicit more detailed feedback. The contextual interview was actually not a separate step in the experiment but was held throughout step 2 and 3, based on the actions and questions of the participants. When interesting opportunities arose after an action or question, the participants were asked to explain why they performed that action or asked that question. From the discussions that followed, a large amount of useful feedback could be extracted.

After filling out the questionnaire, the answers of the participants in that session were compared, analyzed and discussed, especially when they were very different from each other.

VI. RESULTS OF THE FIELD USER STUDY

As discussed in the previous section, we asked the participants (P I, P II and P III) to solve a task and to answer the questions listed in Table III. In a subsequent step, we compared the responses of the participants and tried to bridge differing opinions. In all cases where we identified differing opinions, this was due to a different interpretation of the question. In this section, we discuss the most interesting highlights from the task, questionnaire and anecdotes elicited during the interview.

A. The Assigned Task

All participants could easily solve the assigned task. An interesting result from the process was that participant I and II found additional bottlenecks during the investigation, which were not directly obvious from the rule coverage heat map. From our heat map, it was only obvious that the processor of web server 3 was overloaded. However, because the participants have experience with the EOL infrastructure, they decided to start an investigation for the other web servers as well. The reason for this is that the load of the application is balanced over the web servers and therefore, they wanted to see whether the processor of the other web servers was overloaded as well. As an investigation they compared the deviation-from-mean heat maps of the different web servers to find out if any of them exhibited abnormal behaviour. They found out that all web server processors had the same problem due to a software error. We found out that our approach did not detect a problem on all these servers because it is trained using supervised learning. This means that for a performance problem to be detected and analyzed correctly, a similar case should be in the training set, which is difficult for performance anomalies. This shows that our association rule set either requires better training or a different generation process. We consider this future work.

[7]For a complete list see http://www.st.ewi.tudelft.nl/~corpaul/eol_list.txt

B. Questionnaire Highlights

1) General: The participants' general opinion about WEDJAT was that it is useful and would help them to analyze and diagnose performance problems better and quicker (*Q1, Q2, Q4*, see Table III), even though the usability of the tool has to improve (*Q1*). An example of a usability issue WEDJAT currently has is the following. When the heat maps for a large period of time are being displayed, they are horizontally scrollable. Because the names of the performance counters are on the left side of the heat map, these disappear from the screen as the heat map is being scrolled. While this does not make WEDJAT unusable, it would be more comfortable to have the performance counter names in sight at all times. Improving the usability of WEDJAT by fixing such issues is future work. A promising result is that all participants felt very optimistic about the use of heat maps for performance analysis (*Q3*).

For all views, we asked whether they should be included in a performance dashboard for Exact Online or not. The answers to these questions were different because the participants had different opinions about what the purpose of a performance dashboard should be. Participant III, the product manager, felt that such a dashboard should contain performance analysis tools while the research engineers believed it should contain a trigger to start performance analysis only (*Q10, Q17, Q24, Q31, Q33, Q35*). A reason for this could be that it is easier for the research engineers to access additional performance data when necessary, while this may be more difficult for the product manager. All participants agreed that the intensity graph is the most important starting point for performance maintenance, as it gives the initial indication that the system is performing relatively slow (*Q35*).

In an open question (*Q57*) we asked the participants what their two favourite WEDJAT features were. It is promising to see that all participants liked the new views on the data (rule coverage heat map, deviation-from-mean heat map and the intensity graph) in WEDJAT best. This indicates WEDJAT offers new and useful information.

2) Heat Maps: All participants agreed that the heat map views offer new information which helps them analyze performance problems faster (*Q8, Q9, Q13, Q14, Q20, Q21*). During the discussion we found out that the participants thought the heat maps were also useful for communicating and explaining performance problems to the management and system administrators. An interesting result was that participant I found the heat maps very intuitive, while the other participants were slightly less positive about this (*Q7, Q12, Q19*). From the contextual interview we found out that participant I had a background in statistics, which made him more comfortable with less traditional visualization techniques. An exception to this is the missing values heat map (*Q25, Q26*). Only participant II found this heat map intuitive and useful. The main reason for this is that participant II was the actual participant who requested this feature after the preliminary study. The other participants found this heat map difficult to grasp without extra information, e.g., they would like to receive a 'trigger' when a particular value has been missing, rather than inspect the heat map themselves.

All participants agreed that using the 'traditional' line chart (Section IV-F) and histogram (Section IV-G) in combination with the heat maps improves the usability of WEDJAT (*Q30, Q32*). In addition, they all considered the deviation-from-mean heat maps to be the most useful ones (*Q11-Q16, Q18-Q23*). This is interesting, as our expectation was that the experts would prefer the rule coverage heat map as it gives a direct indication of the bottleneck component. However, during the experiment we found out that the experts preferred to directly view the deviation-from-mean heat maps of servers on which they have seen performance problems occur before, and inspect those for abnormal behaviour (e.g, dark spots in the heat map). If there was no abnormal behaviour on these servers, they referred to the rule coverage heat map to see if it would give another indication of where to look.

3) Applicability: All participants agreed mostly on what they believe WEDJAT is and is not useful for. According to them, WEDJAT is most useful for detecting which server (*Q36*), instance (*Q38*) or hardware (*Q41-Q45*) forms a bottleneck, using the rule coverage and deviation-from-mean heat maps. While they believed that WEDJAT currently is not capable of detecting which load balancer (*Q37*) or network connection (*Q40*) forms a bottleneck, the participants agreed the main cause for this was that we currently did not monitor performance counters for that hardware. The participants expected WEDJAT to be capable of detecting such bottlenecks after the set of monitored performance counters would be extended. We have also asked whether the participants believed WEDJAT could detect which application, e.g., ASP.NET file, forms a bottleneck (*Q39*). All participants agreed WEDJAT was not suitable for this without extra information. We consider detecting application bottlenecks with WEDJAT as future work.

The participants believed the missing values heat map was most useful for detecting hardware failures (*Q47-Q56*). However, the same limitations as with the bottleneck detection apply: load balancer (*Q48*) and network connection (*Q51*) failures are difficult to detect currently, as no performance counters are monitored for them.

C. Additional Insights Obtained From The Interview

In this section we will briefly discuss some of the additional insights that we gained during the contextual interview with the participants. The first insight we obtained was that the participants did not like to cope with graphs that exhibit more than one metric at the same time. For example, the deviation-from-mean heat map used more than one color in the initial version of WEDJAT (green for increasing and red for decreasing values). All participants agreed that this heat map was overly complex and confusing because of the large amount of data. Therefore, we changed the heat map to use one color instead to represent the absolute deviation from the mean, which was appreciated much more by the participants.

The second insight we obtained was that the participants would like a better integration with existing performance tools such as their own dashboard. In addition, they would like to see error log files such as those from the SQL server and web

server integrated with WEDJAT, to display more information for a selected time interval.

Finally, there was some concern with the generation of the association rules and the strength of their diagnosis. It is likely that the generated rules do not cover all performance counters and therefore may give incomplete or insufficient diagnosis of a problem. A possible solution is to offer an additional heat map which gives an overview of performance counters that exhibited abnormal values for a longer period of time. Such a heat map would solve at least part of the problem of supervised learning (see Section VII). Another solution would be to use a continuous training scheme in which the training set is constantly improved through user feedback. The implementation and evaluation of this heat map and a continuous training scheme will be addressed in future work.

VII. DISCUSSION

A. The Research Questions Revisited

1) RQ 1: Does the rule coverage matrix provide a good starting point for performance maintenance?: In our field user study we have seen that the participants appreciated the rule coverage matrix as a starting point for investigating PIOs. The rule coverage heat map was considered useful especially in combination with the deviation-from-mean heat map, which provided some extra information on top of the rule coverage heat map. A possibility to improve the rule coverage matrix is to improve the coverage of the association rule set as discussed in Section VI-A. A final result from the field user study is that users prefer the intensity metric as the initial trigger for starting a PIO investigation, while the rule coverage matrix provides the starting point for pinpointing the bottleneck component.

2) RQ 2: Are heat maps an appropriate way of visualizing performance data?: The results of our field study show that the participants were very enthusiastic about using heat maps for performance maintenance. They all believed using heat maps for performance maintenance will speed up the process. In addition, they were optimistic about using heat maps to explain performance problems to non experts such as hosting companies and product management.

B. Threats to Validity

In this section we discuss the threats to validity of our field user study and our approach for detecting PIOs. For a discussion of the threats to validity of the intensity metric, we refer to [6].

External validity. We have performed our field study on one industrial multi-server SaaS application. Due to its outright scale and set-up, this application is likely to be representative of other large-scale SaaS applications.

Only three participants participated in our field user study, however, all three are performance experts and have many years of experience with performance maintenance.

Internal validity. We use supervised learning to train our association rule set. While this may form a threat to the validity of anomaly detection, we believe this is actually an advantage for the detection of PIOs. Supervised learning implies we will investigate PIOs which occur more often only, making them more interesting for actual performance maintenance.

The complexity of the assigned task might have been too easy. While the task assigned is indeed relatively easy to solve, the main focus of the study was to see whether performance experts appreciated the new view on the performance data. Future work will consist of evaluating WEDJAT using more complex tasks in a controlled experiment setting [11].

A possible threat to validity is the fact that the overhead introduced by monitoring the performance counters influences our training set and therefore our classification scheme. However, as accessing performance counters is relatively cheap, we assume that reading the value of n performance counters will have $O(n)$ overhead for every time period we make a measurement. Because this results in constant overhead for all measurements, we assume that the overhead introduced in the training set will also exist for the measurements made during the classification phase and will therefore be negligible.

Reliability validity. WEDJAT and our implementation for calculating the intensity metric are available for download from our website[8].

C. Lessons Learned

We have learned several lessons from our research. These are the most important:

Performance maintenance is usually done using simple tools while more integrated approaches are desired by the experts (Section I). Experts prefer approaches that can be integrated with their existing performance tools and like to avoid switching between programs and environments as much as possible. *Combining traditional with novel methods for performance maintenance data visualization* can help experts to get comfortable with the novel visualizations quicker (Section VI-B2). *Experts do not like to cope with graphs that exhibit more than one metric at the same time* (Section VI-C). In our effort to combine several metrics in one heat map, we found out that the performance experts did not appreciate this as they found the learning curve for such maps too steep. Instead, they preferred several maps displaying one metric at a time.

It is important not to rely on the rule coverage matrix only when investigating a PIO (Section VI-A) in order to prevent being too dependent on the assocation rule set. It is possible that this rule set is not complete. Instead, it is better to combine the results with those from the deviation-from-mean heat maps to get a broader result.

VIII. RELATED WORK

This section discusses methods for assisting performance experts in finding performance improvement opportunities.

Performance Anomaly Analysis. Important tools for performance experts are anomaly detection mechanisms. Often, these mechanisms detect anomalies that can be prevented in the future by improving the performance of the system.

Breitgand et al. [12] propose an approach for automated performance maintenance by automatically changing thresholds for performance metrics for components, such as response

[8] http://swerl.tudelft.nl/bin/view/Main/MTS

time. In their approach, they set a threshold for the true positive and negative rate of the violation of a binary SLO. Based on this setting, their model tries to predict and adapt the thresholds for components such that the true positive and negative rate converge to their threshold, hence improving the performance of the system. In contrast to our work, they use single threshold values for performance metrics, while we use association rules which lead to combinations of thresholds.

Munawar et al. [13] search for invariants for the relationship between metrics to specify normal behaviour of a multi-tier application. Deviations from this relationship help system administrators to pinpoint the faulty component. In their work they use linear regression to detect relationships between metrics, which limits their research to linear relationships. Our approach does not explicitly look for direct relationships between metrics, but focuses on combinations of values instead.

Cohen et al. [5], [14] present an approach to correlate low-level measurements with SLO violations. They use tree-augmented naive Bayesian networks as a basis for performance diagnosis. Their work is different from ours in the way we detect the possible performance improvement. As we combine several rules, our approach is capable of giving a more detailed analysis of the location of the improvement.

Syer et al. [15] use covariance matrices to detect deviations in thread pools that indicate possible performance problems. The focus of their approach is on thread pools while ours is not limited to a particular architectural pattern.

Malik et al. [16] have presented an approach for narrowing down the set of performance counters that have to be monitored to automatically compare load tests by using statistics. Their technique also ranks the performance counters based on their importance for load tests. Their work focuses on selecting metrics (i.e., the dimension reduction problem), while our work focuses on analyzing those metrics instead.

Jiang et al. [17] analyze log files to see if the results of a new load test deviate from previous ones. This allows developers to analyze the impact of their changes. Nguyen et al. [18] address a similar problem, namely the problem of finding performance regressions. The focus of these approaches is on analyzing whether a change had the desired effect on performance, while our approach focuses on finding what to change.

Profiling. Profilers are tools which collect run-time information about software [8], such as the amount of memory used or the number of instructions executed. More advanced profilers analyze the 'run-time bloat', e.g., unnecessary new object creations [19]. Profilers assist system administrators in the way that they help identify the block or method which uses the most resources and hence may form a bottleneck.

Bergel et al. [20] extend profiling with the possibility to detect opportunities for code optimization. Using visualizations, they advise developers on how to refactor code so that it will run faster. Their advice is based on principles such as making often called functions faster.

In general, while there are methods for decreasing the amount of data and instrumentation [21], [22], execution profiling introduces considerable overhead due to the large amount of data that needs to be monitored. In addition, because profilers usually analyze *hot* code (e.g., the code that uses the most CPU cycles), they are not always directly suitable for detecting all possible performance improvements [22]. Finally, it is possible that many sites must be monitored in a distributed environment. Therefore, while execution profiling plays an important role in performance maintenance, its use should be well-considered and minimally. Our approach can assist in reducing the execution profiling overhead by pinpointing the hardware or server on which profiling should be done.

Using Heat Maps for Performance Maintenance. Heat maps have been used for performance analysis before [23], [24], but we have evaluated our approach in an industrial setting and on multi-server data.

IX. Conclusion

In this paper we have proposed a technique for detecting performance improvement opportunities (PIOs) using association rules and performance counter measurements. We have implemented this technique, together with several novel techniques for the visualization of performance data, in an open source tool called WEDJAT. We have evaluated WEDJAT, the novel visualization methods and our approach in a field user study with three performance experts from industry for a large-scale industrial SaaS application. The results of the user study show that WEDJAT helps them to perform performance maintenance easier and faster. In short, our paper makes the following contributions:

- An approach for using heat maps to analyze the performance of a system and exploit performance improvement opportunities
- The open source tool WEDJAT, which assists during the performance maintenance process
- A field user study in which WEDJAT and the idea of using heat maps for performance analysis are evaluated by three performance experts from industry

In future work we will focus on improving the usability of WEDJAT and the coverage of the used rule set. In addition, we will keep on extending WEDJAT with several new visualization methods. Finally, we will perform a more extended evaluation of our approach in which we will a) do a true/false positive/negative analysis of our results and b) assign more complex tasks to solve to the experts.

Acknowledgement

The authors would like to thank Exact for providing the funds and opportunity to perform this research. Further support came from the *NWO Jacquard ScaleItUp* project.

References

[1] M. Woodside, G. Franks, and D. Petriu, "The future of software performance engineering," in *Future of Softw. Engineering (FOSE)*. IEEE, pp. 171–187.

[2] E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel, "Practical performance models for complex, popular applications," in *Proc. Int'l conf. on Measurement and modeling of computer systems (SIGMETRICS)*. ACM, 2010, pp. 1–12.

[3] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.

[4] R. Berrendorf and H. Ziegler, "PCL – the performance counter library: A common interface to access hardware performance counters on microprocessors," Central Institute for Applied Mathematics – Research Centre Juelich GmbH, Tech. Rep. FZJ-ZAM-IB-9816, 1998.

[5] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in *Proc. of the Symp. on Operating Systems Design & Impl.* USENIX Association, 2004, pp. 231–244.

[6] C.-P. Bezemer and A. Zaidman, "System load characterization using low-level performance measurements," Delft Univ. of Technology, Tech. Rep. TUD-SERG-2012-006, 2012.

[7] L. Wilkinson and M. Friendly, "The history of the cluster heat map," *The American Statistician*, vol. 63, no. 2, pp. 179–184, 2009.

[8] D. E. Knuth, "An empirical study of fortran programs," *Software: Practice and Experience*, vol. 1, no. 2, pp. 105–133, 1971.

[9] C.-P. Bezemer and A. Zaidman, "Multi-tenant saas applications: maintenance dream or nightmare?" in *Proc. of the Joint ERCIM Workshop on Software Evolution (EVOL) and Int. Workshop on Principles of Software Evolution (IWPSE)*. ACM, 2010, pp. 88–92.

[10] K. Holtzblatt and S. Jones, "Human-computer interaction." Morgan Kaufmann, 1995, ch. Conducting and analyzing a contextual interview (excerpt), pp. 241–253.

[11] B. Cornelissen, A. Zaidman, and A. van Deursen, "A controlled experiment for program comprehension through trace visualization," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 341–355, 2011.

[12] D. Breitgand, E. Henis, and O. Shehory, "Automated and adaptive threshold setting: Enabling technology for autonomy and self-management," in *Proc. Int. Conf. on Autonomic Computing (ICAC)*. IEEE, 2005, pp. 204 –215.

[13] M. A. Munawar, M. Jiang, and P. A. S. Ward, "Monitoring multi-tier clustered systems with invariant metric relationships," in *Proc. Int'l Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM, 2008, pp. 73–80.

[14] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox, "Ensembles of models for automated diagnosis of system performance problems," in *Proc. Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 2005, pp. 644 – 653.

[15] M. Syer, B. Adams, and A. Hassan, "Identifying performance deviations in thread pools," in *Proc. Int'l Conf. Softw. Maintenance (ICSM)*. IEEE, 2011, pp. 83–92.

[16] H. Malik, Z. M. Jiang, B. Adams, A. Hassan, P. Flora, and G. Hamann, "Automatic comparison of load tests to support the performance analysis of large enterprise systems," in *Proc. Conf. on Softw. Maintenance and Reengineering (CSMR)*. IEEE, pp. 222–231.

[17] Z. M. Jiang, A. Hassan, G. Hamann, and P. Flora, "Automated performance analysis of load tests," in *Prof. Int'l Conf. Softw. Maintenance (ICSM)*. IEEE, pp. 125–134.

[18] T. H. Nguyen, B. Adams, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Automated detection of performance regressions using statistical process control techniques," in *Proc. joint WOSP/SIPEW Int'l Conf. on Performance Engineering (ICPE)*. ACM, pp. 299–310.

[19] D. Yan, G. Xu, and A. Rountev, "Uncovering performance problems in java applications with reference propagation profiling," in *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE CS, 2012, pp. 134–144.

[20] A. Bergel, F. Baados, R. Robbes, and W. Binder, "Execution profiling blueprints," *Software: Practice and Experience*, *To appear*.

[21] S. Elbaum and M. Diep, "Profiling deployed software: assessing strategies and testing opportunities," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 312–327, 2005.

[22] M. Jovic, A. Adamoli, and M. Hauswirth, "Catch me if you can: performance bug detection in the wild," in *Proc. Int'l Conf. on Object oriented programming systems languages and applications (OOPSLA)*. ACM, 2011, pp. 155–170.

[23] K. Fürlinger, M. Gerndt, and J. Dongarra, "On using incremental profiling for the performance analysis of shared memory parallel applications," in *Proc. Int'l Euro-Par Conference*, ser. LNCS, vol. 4641. Springer, 2007, pp. 62–71.

[24] B. Gregg, "Visualizing system latency," *Commun. ACM*, vol. 53, no. 7, pp. 48–54, 2010.

TABLE III
QUESTIONNAIRE RESULTS

		P I	P II	P III
General (1 = strongly disagree, 5 = strongly agree)				
Q1	The tool was easy to use.	4	3	2
Q2	A tool like Wedjat will save me time.	5	3	5
Q3	There is added value in using heatmaps for performance analysis.	5	4	5
Q4	A tool like Wedjat will help me understand and diagnose performance problems better.	4	4	5
Q5	Switching between different datasets in Wedjat was easy.	4	3	4
Rule coverage heat map				
Q6	The meaning of this heatmap was clear to me.	4	3	3
Q7	The heatmap was intuitive.	4	3	3
Q8	The heatmap provides new information.	4	3	4
Q9	The heatmap improves the speed with which problems can be analyzed.	4	3	4
Q10	Adding the rules heatmap view to a performance dashboard for EOL will save me time.	4	1	5
Deviation-from-mean heat map				
Q11	The meaning of this heatmap was clear to me.	4	3	3
Q12	The heatmap was intuitive.	5	3	3
Q13	The heatmap provides new information.	4	3	4
Q14	The heatmap improves the speed with which problems can be analyzed.	4	3	4
Q15	I clicked in the heatmap to get more details about a value.	4	4	5
Q16	The extra information revealed after clicking was useful.	5	4	5
Q17	Adding the raw values heatmap view to a performance dashboard for EOL will save me time.	2	1	4
Deviation-from-mean heat map per server				
Q18	The meaning of this heatmap was clear to me.	4	3	3
Q19	The heatmap was intuitive.	4	3	3
Q20	The heatmap provides new information.	3	3	4
Q21	The heatmap improves the speed with which problems can be analyzed.	3	3	4
Q22	I clicked in the heatmap to get more details about a value.	4	3	5
Q23	The extra information revealed after clicking was useful.	4	3	5
Q24	Adding the raw values per server heatmap view to a performance dashboard for EOL will save me time.	3	1	4
Missing values heat map				
Q25	The missing values tab was intuitive.	2	4	3
Q26	The missing values tab was useful.	2	4	2
Line chart				
Q27	The meaning of this chart was clear to me.	5	4	5
Q28	The chart was intuitive.	5	4	5
Q29	The chart provides new information.	4	4	3
Q30	The chart improves the speed with which problems can be analyzed.	4	4	5
Q31	Adding the line chart to a performance dashboard for EOL will save me time.	3	1	4
Histogram				
Q32	I used the extra information provided in the histogram tab.	4	3	5
Q33	Adding the histogram tab to a performance dashboard for EOL will save me time.	2	1	3
Intensity				
Q34	I used the extra information provided in the intensity tab.	4	3	5
Q35	Adding the intensity tab to a performance dashboard for EOL will save me time.	4	5	4
Applicability: bottleneck detection (1 = WEDJAT is very unuseful for this, 5 = WEDJAT is very useful for this)				
Q36	Detecting the server that forms the bottleneck.	5	5	5
Q37	Detecting the load balancer that forms the bottleneck.	3	1	1
Q38	Detecting the instance that forms the bottleneck.	5	4	5
Q39	Detecting the ASPX that forms the bottleneck.	1	1	1
Q40	Detecting the network connection that forms the bottleneck.	1	1	1
Q41	Detecting the hardware that forms the bottleneck.	5	4	3
Q42	Detecting the CPU that forms the bottleneck.	5	4	5
Q43	Detecting the HDD that forms the bottleneck.	5	4	5
Q44	Detecting the memory that forms the bottleneck.	5	4	5
Q45	Detecting other hardware that forms the bottleneck.	5	1	3
Q46	Detecting a thread pool that forms the bottleneck.	1	3	1
Applicability: failure detection				
Q47	Detecting a server failure.	4	3	5
Q48	Detecting a load balancer failure.	3	1	1
Q49	Detecting an instance failure.	4	3	5
Q50	Detecting an ASPX failure.	1	1	1
Q51	Detecting a network failure.	3	1	4
Q52	Detecting hardware failure.	4	3	5
Q53	Detecting a CPU failure.	4	3	5
Q54	Detecting a HDD failure.	4	3	5
Q55	Detecting a memory failure.	4	3	5
Q56	Detecting other hardware failures.	3	1	4
Q57	**Favorite features in WEDJAT:**			
PI	Deviation-from mean heat map, intensity graph			
PII	Rule coverage heat map, intensity graph			
PIII	Rule coverage heat map, deviation-from mean heat map			

TUD-SERG-2012-013
ISSN 1872-5392