# The Performance Mining Method:

# Extracting Performance Knowledge from Software Operation Data

*Stella Pachidi & Marco Spruit*
Department of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
Emails: s.pachidi@vu.nl & m.r.spruit@uu.nl

## Abstract

Software Performance is a critical aspect for all software products. In terms of Software Operation Knowledge, it concerns knowledge about the software product's performance when it is used by the end-users. In this paper we suggest data mining techniques that can be used to analyze software operation data in order to extract knowledge about the performance of a software product when it operates in the field. Focusing on Software-as-a-Service applications, we present the Performance Mining Method to guide the process of performance monitoring (in terms of device demands and responsiveness) and analysis (finding the causes of the identified performance anomalies). The method has been evaluated through a prototype which was implemented for the main online financial management application in the Netherlands.

**Keywords***:* knowledge management, software-as-a-service, data mining, software performance, software maintenance, software operation knowledge

## Introduction

As the exponential increase of information is forcing us into the era of Big Data (Demirkan & Delen, 2013) and the organizations of the 21st century function in a global marketplace defined by intense and growing turbulence (Heinrichs & Lim, 2003), the need for tools that enable quick and effective extraction of business insights is more evident than ever. Business Analytics has evolved and includes techniques to access, report and analyze data to understand, analyze and forecast business performance (Delen, & Demirkan, 2013). Such techniques include data mining, which has been used to support knowledge workers in generating relevant insights (Heinrichs & Lim, 2003) to support decision making in various processes at different strategic levels (Bolloju, Khalifa, & Turban, 2002; Courtney, 2001).

Knowledge Management has been recognized as an important process in software organizations for supporting core software engineering activities in order to decrease costs, increase quality and lead to better decisions (Bjørnson & Dingsøyr, 2008; Lindvall & Rus, 2002). Although a lot of research has focused on managing knowledge in software development (Moreno García et al, 2004), little attention has been drawn to the management of knowledge on the customers' experience with deployed software products (van der Schuur, Jansen, & Brinkkemper, 2010) for software maintenance purposes (Midha & Bhattacherjee, 2012). However, the rise of cloud computing (Marston et al., 2011) designates the need of Software-as-a-Service (SaaS) organizations to extract and analyze knowledge on how their software operates (van der Schuur et al., 2010), in order to efficiently manage with unexpected performance issues (Zo, Nazareth & Jain, 2010) and increasing scalability (Delen, & Demirkan, 2013), to maintain quality of their services (Sun, He & Leu, 2007). With this paper we aim to contribute to the management of knowledge related to software operation (van der Schuur et al., 2010), with the goal to support software maintenance processes (Midha & Bhattacherjee, 2012). Specifically, we answer the research question: *How can we detect performance anomalies and their causes in software operation data?* We contribute to the domain of performance evaluation by following a meta-algorithmic approach, which entails extracting new knowledge through the reuse of already proven algorithms in new domains of application. Thus, we examine several data mining techniques applicable for the extraction of performance knowledge from software operation data, and construct a method that incorporates different techniques and helps structure the complicated process of performance mining. We evaluate the method and the selected data mining techniques by means of a prototype that was run with operation data of an established online financial management application.

## Software Performance Evaluation

Performance is described by the proportion of the amount of effective work that is accomplished by a software system, over the time and resources used, in order to carry out this work (Zhu, 2014). In practice, software performance may be characterized by different aspects, which are evaluated in measurable terms using different performance metrics, such as response time, throughput, availability, latency, or utilization of resources (e.g. percentage of CPU or memory usage). Software Performance Evaluation is a critical process for all types of software products, indispensable at every stage of the product's life (Zhu, 2014). It is performed to determine that the system meets the user-defined performance goals and detect possible

improvement points; to compare a number of alternative designs and select the best design; or to compare a number of different solutions and select the system that is most appropriate for a given set of applications (Jain, 2008).

A common technique for evaluating software performance is monitoring the system, while it is being subjected to a specific workload (Jain, 2008). Monitoring consists in observing the performance of systems, collecting performance statistics, analyzing the data and displaying the results. The commonest functionalities of a monitor include: finding the most frequently accessed software segments, measuring the utilization of resources, finding the performance bottlenecks, etc. A performance monitor can provide valuable information about the application and system run-time behavior, in order to carry out a dynamic analysis of the system's performance.

**Extracting Performance Knowledge from Software Operation Data**

As the amount of data that software vendors have to process is raising exponentially, efficient extraction of knowledge from these data is indispensable. Knowledge on software performance (in terms of device demands or responsiveness) may be elicited from data gathered while the software product is used by its end-users (software operation data) (van der Schuur et al., 2010). So far research has focused on the recording of software operation data, and on the integration, presentation and utilization of the knowledge acquired through general statistics and visualization techniques (van der Schuur et al., 2010). However, sufficient support of data analysis functionalities is missing.

Data mining techniques could be used to extract software performance knowledge from operation data: we could inspect the duration of an operation until it is fulfilled and get an alert when everything starts to slow down; identify when the capacity of resources is limited; find deadlocks and query time-outs in the database; calculate the throughput of processing requests on a web server, etc. By monitoring the performance on a regular basis (daily, weekly, monthly), we are also able to compare the values of the metrics and find the causes of performance problems: there might be a hardware problem on a server, or overuse from a specific customer, a hardware or bandwidth problem on the end-user's side, or a software bug on a release update that may incur extra computation time, and other things.

Performance monitoring and analysis through software operation data mining (*e.g.* Pachidi, Spruit & Weerd, 2014) could contribute significantly to the performance testing and quality control processes, as it can provide an automated alert system that identifies problems on the performance of a software product and even analyzes their cause. The direct advantage would be the increase of quality of the product, which would also help increase the customer satisfaction (McKinney, Yoon, & Zahedi, 2002). Another advantage would also be a decrease in the costs for performance testing, since many problems would be identified, or even prevented, through the automated analysis of the software operation data.

Although research has been performed to some extent on the use of data mining techniques for performance monitoring (Jain, 2008; Vilalta et al., 2002), most of these approaches are used to test performance models or for simulations, rather than to analyze how the software actually performs in the field. The approaches that do so, focus on one layer only (e.g. network behavior) and do not cover all possible tiers. Also, most techniques try to produce alerts for performance bottlenecks (Imberman, 2001), and are not sufficient for pointing out the

problem causes. Approaches that use data mining techniques to monitor in-the-field performance (Imberman, 2001), focus on one tier only (Borzemski, 2009), or they are used for producing alerts and not for pinpointing the problem causes. We still need a uniform way of analyzing software operation data, in order to monitor and analyze the performance of software products in the field. This paper aims to cover this gap by exploring how data mining techniques can be performed on software operation data, in order to 1) detect anomalies in the performance of the software on the end-user's side, and 2) reason about the cause of each identified anomaly.

**Related Work**

Performance monitors have been used for several purposes, such as capacity planning (Thakkar et al., 2008), performance tuning (Musumeci & Loukides, 2002), workload characterization (Avritzer et al., 2002), and program profiling (Ammons, Ball, & Larus, 1997). An example methodology that uses monitors is NetLogger (Gunter & Tierney, 2003), which implements real-time diagnosis of performance problems in complex high-performance distributed systems through the generation of time stamped event logs.

Some preliminary research on testing performance with log file analysis was published by Andrews and Zhang (2003), who suggested a framework for finding software bugs and testing the system performance, by tracking for example internal implementation details (e.g. memory allocation). Johnson et al. (2007) also incorporated the generation of performance log files in the performance testing process, to enable developers and performance architects to examine the logs and identify performance problems. Xu et al. (2009) suggest mining techniques for analyzing console logs in order to detect problems in large-scale systems such as online games. The Microsoft Application Consulting and Engineering (ACE) Team (2003) have built the System Monitor: a process that monitors hardware and software resources in a machine for the real-time monitoring of performance counters -metrics for hardware and software resources. The System Monitor logs the counter information for later analysis, triggers alerts when performance thresholds occur, and generates trace logs which record data (processes, threads, file details, page faults, disk I/O, network I/O, etc.) when certain activities occur.

As far as data mining is involved in the performance analysis task, some related research has been conducted. Imberman (2001) identifies the parts of the Knowledge Discovery in Databases process (Maimon & Rokach, 2005), which are currently being followed by performance analysts: regression, graphical analysis, visualizations and cluster analysis. She also explains further the parts that are less often used so far, but could be useful for analyzing performance data (association/dependency rule algorithms, decision tree algorithms and neural networks). Jain (2008) suggests the regression technique, for modeling performance related variables, and time series analysis, for managing and analyzing performance data gathered over a time period. Vilalta et al. (2002) study the use of predictive algorithms in computer systems management for the prediction and prevention of potential failures. Pray and Ruiz (2005) present an extension of the Apriori algorithm for mining expressive temporal relationships from complex data, and test it with a computer performance data set. Finally, Warren et al. (2010) suggest an approach for finding the "hot spots" of a program, i.e. the operations that have the highest impact on performance.

## Research Methodology

In this research we employ a meta-algorithmic approach, by incorporating state-of-the-art data mining techniques in a method. We follow the Design Science Research (DSR) paradigm (Hevner et al., 2004): questions are answered through the creation and evaluation of innovative artifacts that render scientific knowledge and are fundamental and useful in understanding and solving the related organizational problem (Hevner & Chatterjee, 2010). In our case, the research artifacts correspond to the method and to the prototype (which actually constitutes an instantiation of a method) for software performance mining. We construct the method for performance mining by using the Method Engineering approach provided by van de Weerd and Brinkkemper (2009), as previously introduced in this journal by Vleugel, Spruit & Daal (2010). To evaluate the effectiveness and applicability of the method, we perform case study research in a software company.

This research is delimited by some specific constraints. First of all, we limit our research to the analysis of performance for software-as-a-service applications. However, the extensibility to other types of product settings will be considered in the construction of our method. Furthermore, concerning the acquisition of software operation data, we assume that they are gathered through logging tools and libraries, which are embedded in the software product's code. Thus, all actions performed by the end-users are recorded and stored in a central database. Finally, we have selected to work with the R environment end programming knowledge (Venables et al, 2014). This decision influences our research on the selection of techniques that will be implemented in the method and the prototype.

In order to structure our data mining research, but also to assemble our Performance Mining method, we follow the CRISP-DM Reference Model (Chapman et al, 2000). The CRoss Industry Standard Process for Data Mining consists of six phases, each of which constitutes a classification of data mining activities on the high level, and the information flows between them. In this paper, the Business Understanding phase may be mapped to the goals of performance analysis presented in the Introduction as well as to the hypotheses for performance analysis that are set in section 4. The Data Understanding phase includes the identification of performance variables in section 5. The Modeling phase corresponds to the data mining techniques presented in section 6. The Evaluation phase may be mapped to section 8, where the evaluation by means of a case study is presented. Part of the Deployment phase may be mapped to the discussion in section 9. Even though the Data Preparation phase was a significant part of this research, it is not thoroughly discussed in the current paper.

## Hypotheses for Performance Analysis

As the aspect of performance is a very complicated and wide domain, it is necessary to set some hypotheses about what types of performance problems we aim to detect and analyze with our suggested techniques and method. As we also mentioned in the introduction, this paper presents research on performance monitoring and analysis of software-as-a-service applications. Logging techniques are used to collect the performance data during the utilization of the software product in the field.

The performance anomalies that we aim to inspect in software operation data can be

categorized into two types: problems related to device demands on the servers, which include unusually high utilization of resources (CPU, memory, disk, etc.), in the computers on which the servers are running; and problems related to responsiveness, which refer to situations when the response time (i.e. the time interval between the end of a request submission by the user and the end of the corresponding response from the system) is much higher than usual.

Concerning responsiveness problems, we consider the following possible causes that could be responsible for increasing the response time:

- A problem on the server (e.g. a memory leak, a processor problem, etc.) delays all operations running on that server.
- The responsiveness of a server will be lower when the load is too high during particular, repetitive time intervals of the day/week/month (seasonal effect).
- There is a software problem (e.g. a software bug) with a specific application, function, action, etc. which delays the respective response time.
- There is a processing problem (e.g. parsing problem) that delays the processing time (Processing duration = Total response time Query Duration).
- A problem on the database (e.g. a deadlock or a query time-out) delays the duration of a query execution.
- The size of data from specific customers might be substantially high, compared to the usual size, thus the response time to view, edit, store, etc. the related records can be much longer.
- A problem on the user's side (e.g. too heavy customization of a menu, or a bandwidth problem) might delay the duration of the applications, functions, etc. that he/she is using.

The aforementioned hypotheses create a specific context, in which we search for the appropriate data mining techniques and we construct the performance mining method and the prototype.

## Performance Variables

In order to gather the software operation data that are necessary to monitor and analyze performance, we need to decide which variables will need to be inspected. We distinguish three categories of variables: a) performance metrics related to the resources usage on the machines where the servers are running; b) performance metrics related to responsiveness; c) variables related to the software operation details.

Resources usage variables include performance metrics, related to the resources used on the computers on which the servers that host the software-as-a-service product are running. These variables are useful to detect bottlenecks located on a specific server, which decrease the performance of the product. The metrics are classified according to the resource they refer to (Team A. C. E., 2003): processor, system object, physical disk, memory, network interface, web tier and SQL tier.

Responsiveness variables concern how fast a request, action etc. is processed by a server. We suggest three variables that can be used to measure the responsiveness of the software product: duration, which refers to the response time, i.e. the total time elapsed (in milliseconds) from the moment the end-user finishes a request (e.g. request to load a .aspx page) until the server completes the response to that request; query duration, which is part of the total duration and describes the time elapsed to process and respond to a query request; and throughput, which refers to the rate of requests that are processed by the server in a sample time interval (usually one second).

Operation details variables provide information about each action that is performed on the software product by an end-user: who is using the software product (customer, user ID, IP address), where the application is being hosted (web server, database), which functionalities are accessed (specific application, page, function, method, etc.), how (which specific action e.g. "new", "edit", "save", etc.), when (date and time, session, etc.), which background tasks are running at the same time, which errors appear, and other details (e.g. how many records are loaded or stored, and so forth).

## Data Mining for Performance Analysis

In this section we present data mining techniques that may be applied on software operation data, in order to analyze the performance of software-as-a-service products. We suggest three main performance analysis tasks: Monitoring performance on the servers is related to monitoring the resources utilization on the machines where the servers are running. We are interested in detecting performance bottlenecks related to the resources usage, and we aim to provide an estimation of the future capacity of the resources. Monitoring performance of applications consists in monitoring the responsiveness of the system, with regard to individual applications (e.g. .aspx pages, or methods, etc.). We are interested in detecting delays in the response time of applications, or even predicting possible decrease of the responsiveness given the current configuration. Analyzing performance on applications aims to find the causes (as presented in section 4) for events where response time is significantly higher than usual.

### Monitoring Performance on the Servers

In order to monitor the utilization of resources on the server machines, values from the metrics presented in section 5 should be collected periodically (between time intervals of e.g. 1-3 minutes). Each metric value will be stamped with the specific date and time of its measurement; hence we get to have a sequence of values, which can be considered as time-series data (Han, 2005).

The treatment of all performance metrics variables as time-series data will be the same: each performance counter is a time series variable, which can be viewed as a function of time t : $Y=Y(t)$ and can be illustrated as a time-series graph (Han, 2005). We will apply time series analysis and burst detection to model the time series, and subsequently trend analysis, to forecast future values of the time-series variable.

Time series analysis consists in analyzing a time series into the following set of features,

which characterize the time series data (Cowpertwait & Metcalfe, 2009; Han, 2005): Trend or long-term movements are systematic changes in a time series, which do not appear to be periodic, and indicate a general direction (e.g. a linear increase or decrease), in which a graph is moving over a long period of time. We are interested in identifying the trends, in order to identify when there is a long-term increase or decrease of a performance counter, which could give insight into a performance problem. Cyclic movements or cyclic variations are long-term oscillations of a trend line or curve, and may appear periodically or aperiodically. Seasonal movements or seasonal variations correspond to systematically repeating patterns that a time series appears to follow within any fixed period (e.g. increased usage every Monday morning). Irregular or random movements indicate the occasional motion of a time series because of random or chance events. An example of a time-series decomposition may be viewed in figure 2.

Apart from reflecting interesting patterns and features of the data, a time series plot may also expose "burstiness", which implies that more significant events are happening within the same time frame (Vlachos et al., 2008) and therefore may provide awareness for an impendent change of the monitoring quantity, so that the system analyst can proactively deal with it. Burst detection is necessary in order to remove the bias of the burst intervals in the trend analysis. The burst intervals will be replaced with new values through linear interpolation and the new values of the data points will correspond to the respective mean value of the moving average. The time series will then be decomposed again, in order to get the updated trend.

When monitoring performance counters we want to receive an alert when significant increases or decreases of the performance counters are expected, in order to prevent potential performance problems. In the context of forecasting the time series of performance counters, we suggest following trend analysis, which includes the extraction of the general direction (trend) of a time series, the analysis of the underlying causes which are responsible for these patterns of behavior, and the prediction of future events in the time series based on the past data. We use the technique of trend estimation (Han, 2005), which fits a linear model in the trend curve of the time series through linear regression, and subsequently performs extrapolation, i.e. constructing new data points based on the known data points. The fitted line helps us quickly and automatically get conclusions about possible significant increase or decrease of the counter being monitored. An estimation of the percentage of the increase/decrease can also be calculated using the equation of the predictor variable. This technique provides the stakeholders with an automated way to identify and study significant increases or decreases of the performance counters, as well as predict and thus prevent future performance problems.

**Monitoring the Performance of Applications**

In order to monitor the responsiveness of the system, we need to collect information every time the end-user performs an action on the software product. Every action that happens on the end-user's side is represented by a record that includes a date and time, details of the operation and the response times. Therefore, the responsiveness data are sequence data, since they constitute sequences of ordered events, and can be treated as time-series data.

However, the data related to the performance of individual applications, differ from the time-series data that we studied in the previous task: here, we have sequences of values which are measured at unequal time intervals. Furthermore, the same application may be accessed by

more than one users at the same time (e.g. running on different servers). Therefore, although we propose again time series analysis, burst detection and trend analysis, their implementation will be different. More specifically, we need to select a sufficient time period (e.g. one hour) within which most software applications are expected to have been used by the end-users. Also, we choose to omit hours (e.g. between 00:00 and 06:00) or days (e.g. weekends), if the software product has too low usage during these time intervals.

The trend analysis will help detect an important increase of the response times of an application, which needs further inspection and analysis to identify the cause of that increase. The seasonal variations curve informs when an application is over-used by the end-users, when the system responsiveness is lower, etc. The bursts detection will identify time intervals with response times significantly higher than usual, which should be further inspected and removed from the time series.

**Analyzing the Performance of Applications**

When the response time of the system to process a request from an end-user is significantly higher than usual, we consider that we have a performance problem. Several reasons may lie behind the delaying of performance (e.g. a problem on the server). In this research, we consider the causes described in section 4. Since the reasons of performance problems vary, we will need to use as much information as possible, thus all variables presented in section 5 are relevant to this analysis task.

Association rules mining is a popular data mining technique, which is used to discover interesting relationships that are hidden in large data sets. The relationships constitute relations between attributes on the value-level, which are represented by association rules, i.e. expressions in the form of $A \rightarrow C$ , where A and C are sets of items in a transactional database, and the right arrow implies that transactions which include the itemset A tend to include also the itemset C (Agrawal, Imielinski, & Swami, 1993). The problem of association rules mining on performance data consists of searching for interesting and significant rules that describe relationships between attributes which expose performance problems (i.e. high response times).

In order to select interesting rules, there are several measures of interestingness, among which the commonest are the measures of support and confidence (Agrawal & Srikant, 1994): The support supp(X) of a rule $X \rightarrow Y$ is defined as the proportion of transactions in the database that contain both itemsets X and Y, over the total number of the transactions in the database. The confidence of a rule $X \rightarrow Y$ is defined as the conditional probability of having the itemset Y given the itemset X; in other words, the proportion of transactions in the database that contain both itemsets X and Y, over the number of the transactions which contain the itemset X. Consequently, the problem of association rules mining could be defined as the discovery of interesting rules, i.e. rules that outweigh the thresholds we set for the minimum support and the minimum confidence (Tan, Steinbach & Kumar, 2005).

Frequent pattern mining constitutes the discovery of sets of items in a database that appear frequently in the same transactions, i.e. with support measure higher than the minimum support threshold (Han et al., 2007). The computational complexity of frequent itemset generation could be reduced by the A Priori principle, according to which “a set of items can

only be frequent if all its (non-empty) subsets are frequent" (Agrawal & Srikant, 1994). Also, the number of comparisons can be reduced by compressing the dataset or using advanced data structures (Tan et al., 2005). After mining all frequent itemsets, we can generate all possible rules that have confidence higher than the minimum confidence threshold. The resulting rules may be post-processed, i.e. ordered by their measures of interestingness, such as the lift measure, measures whether the transactions that contain itemset X actually have an increased probability of having itemset Y compared to the probability of all transactions of the database.

In this section we reviewed the data mining techniques which may be performed on performance data of software-as-a-service applications, for detecting performance anomalies, as well as reasoning about their causes. An overview is provided in table 1.

Table 1

*Summary of Data Mining Techniques for Performance Monitoring and Analysis*

| **Analysis Task** | **Performance Type** | **Data Mining Techniques** |
|---|---|---|
| Monitoring performance on the servers | Device demands | Time series analysis |
| | | Burst detection |
| | | Trend analysis |
| Monitoring performance of applications | Responsiveness | Time series analysis |
| | | Burst detection |
| | | Trend analysis |
| Analyzing performance of applications | Causes of performance anomalies | Association rule mining |

## Performance Mining Method

In order to provide guidance to the software vendors in the analysis and monitoring of their software products' performance, we propose the Performance Mining Method. The method has been constructed using the Method Engineering approach as described by van de Weerd and Brinkkemper (2009). The method is aimed to be used for mining performance and usage data of software-as-a-service applications, which are collected in a central point on the software vendor's side. However, it could easily be adjusted-expanded for other types of products. In the method we only included a subset of the aforementioned data mining techniques, which includes the techniques employed in the Performance Mining Prototype. It would be an interesting piece of future research to experiment with the remaining suggested techniques, and then include in the method the possibility to select which data mining technique will be used.

The method includes three activities, which correspond to the three suggested tasks: monitoring the performance on the servers, monitoring performance of applications, and analyzing performance of applications. Each activity consists of several steps (sub-activities), which follow the CRISP-DM cycle (Chapman et al, 2000), starting from the business understanding phase, and following the phases of data understanding, data preparation,

modeling, until the evaluation phase. The whole Process Deliverable Diagram is not provided here, because of its size and complexity. Instead, the process diagram is presented in figure 1.

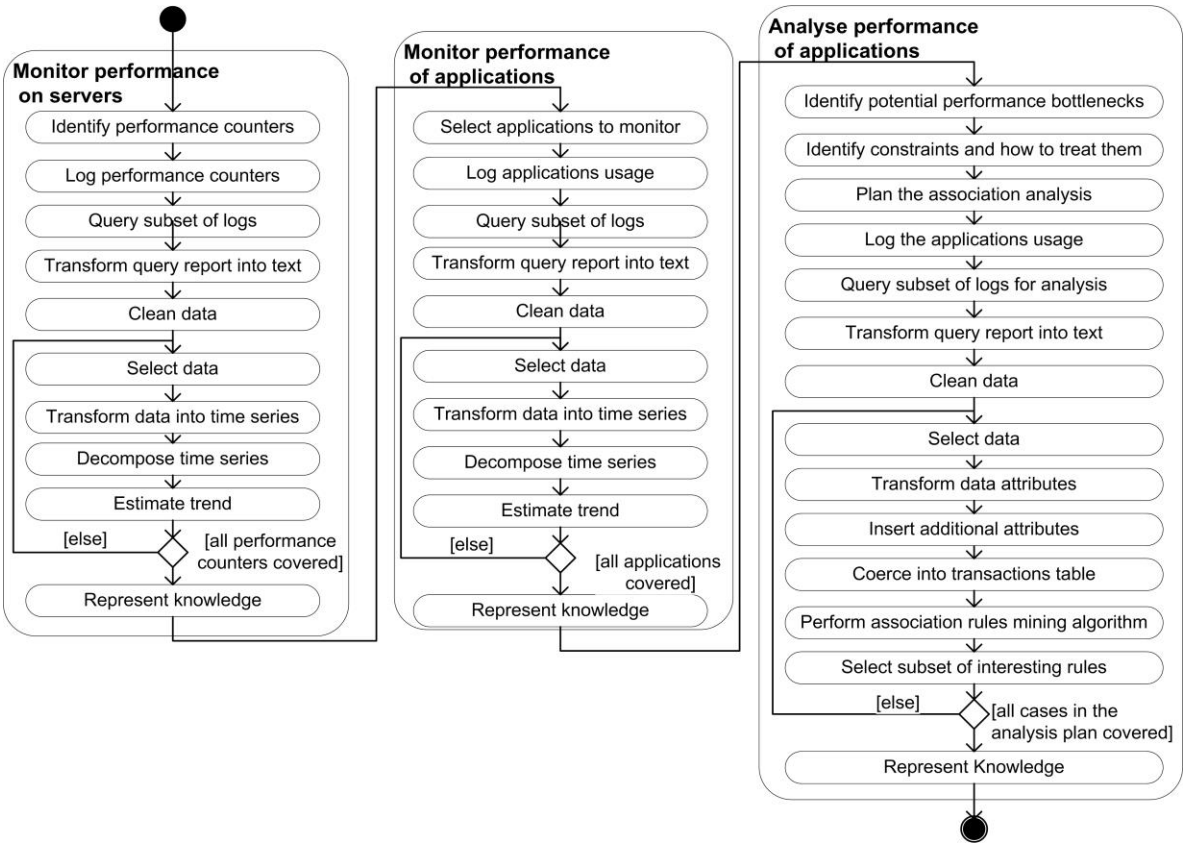


*Figure 1*. Process Diagram of the Performance Mining Method

## Evaluation of the Performance Mining Method

The Performance Mining Method is instantiated in a prototype, which has been created in order to implement and validate the method and the suggested data mining techniques. In order to validate our method, we performed a case study in a Dutch software company, where we implemented the Performance Mining Method and ran the prototype in the context of a real software product.

### Performance Mining Prototype

The Performance Mining Prototype is implemented in the programming language and environment R (Team R. C., 2012), and is aimed to analyze usage and user data which are gathered through logging tools, and are stored in logs, in a central database. The prototype consists of three scripts which correspond to the three main analysis tasks: monitoring performance on the servers, monitoring performance of applications, analyzing performance on applications. Furthermore, one function has been implemented in Java, with the task to prepare the log file in a format suitable for importing in R.

For the task of monitoring performance on the servers, we created a function that is used

to estimate the trend of each performance counter, after having pre-processed the data (following the steps that were presented in the respective activity of the Performance Mining Method). The function creates the time series object, performs some extra preprocessing (e.g. handling the missing values), performs time series analysis, calculates the moving average and identifies the burst intervals, and then removes the bursts, and finally estimates the trend through linear regression. All outputs from the analysis are printed in image files, which are stored in the previously defined output folder.

The script for the task of monitoring performance of applications includes time series decomposition, burst detection and trend estimation. Thus, it is very similar to script that was created for monitoring performance on the servers. However, here we do not have measurements that are collected periodically 24×7. The solution to overcome this complication is to select a sufficient time period (e.g. one hour) within which most software applications are expected to have been used by the end-users. Also, it might make sense to omit hours (e.g. night time) or days (e.g. weekends), if the software product has too low usage during these time intervals. A function with similar functions as the aforementioned one is implemented in this task for the estimation of the trend for the response time of each application.

In the analysis of performance of applications, we try to detect the causes for performance problems (i.e. too long duration of an application) that occur during software operation. We use Associations Rules Mining through Frequent Pattern Mining and search for interesting and significant rules that describe relationships between operation attributes which expose performance problems. We use the Apriori algorithm (Hahsler et al., 2009). We set as parameters for minimum support threshold minsup = 0.005 and for minimum confidence minconf = 0.6. Although especially the support threshold may seem too low, we should remind here that the performance problems that we try to analyze constitute very unique and rare cases, especially if we consider the variety of users who may use the product, and the variety of applications that are inspected. However, these parameters can and should be changed and adjusted to the specifications of the product that is under study each time.

## Case Study

In order to validate the Performance Mining Method and Prototype, we performed a case study in a Dutch software organization that delivers business software solutions to more than 100,000 small to medium enterprises. Our case study was performed in the context of an internet-based accounting solution which constitutes one of the main software products of the company. This Software-as-a-Service application serves over 10.000 customers and is used by more than 3.500 users per day.

The application logs performance and usage data, through a log-generating code that is incorporated in the system layer. Every time the application is used by an end-user, several variables are stored in log tables in the administration database, on the server's side. The information that is logged contains: usage data (user id, application used, date/time, action taken, etc.), performance data (how long it took to process a query, how long it took to load an application, etc.), quality data (errors that appeared during usage, etc.) and other useful information such as which background tasks were running during usage. All logs are stored in the database for the period of 90 days. Furthermore, the System Monitor is also used to log

counters related to usage of resources in the machines where the servers (Microsoft Windows Servers) are running. However, no sophisticated analysis is performed in any of the logs, and the analysts only perform basic statistical operations on the data and produce common statistical plots for their monitoring.

In the first months of year 2010, the application faced several performance issues, as a lot of new customers started working with the product. Similarly to a traffic jam, the load on the web servers, network, SQL server etc. increased significantly and resulted in clogging the system and eventually bringing it to a standstill. In numbers, the performance analysts observed an increase of the average time of processing web requests from about 350ms to about 500ms.

We tried to observe how our suggested Performance Mining method could be applied for this case, in order to monitor its performance and analyze the performance problems that appear. We also ran our prototype with sample log data collected during the product's operation, in order to validate its functionality. In the following paragraphs we present the result of each performance mining task.

**Monitoring Performance of Servers**

We selected metrics related to categories of resources that are used on the machines where the web servers are running: processor, system object, the physical disks, the memory, network interface, the web tier, and the SQL tier. We used the related script from our prototype to monitor performance counters that were measured on the machines of two web servers in the period of two months, from 1st February 2010 until 1st April 2010. We used the prototype to pre-process the dataset (transforming, cleaning, selecting data, etc.), create the time series object, and perform time series decomposition, burst detection and trend estimation. In the end, the knowledge was represented in the format of images, which could be inspected by the performance analysts.

An example of the output can be viewed in figure 2, which illustrates the time series decomposition for the performance counter %Processor Time on server $x_1$. In the observed data plot we can clearly identify a burst interval between day 13 and 15. Such a burst would influence the result of our trend analysis and therefore needs to be handled before estimating the trend of the counter. Furthermore, the burst raises a lot of concern to the performance analyst, who will need to conduct a thorough analysis of the logs recorded in that time interval, in order to look for the causes of these anomalous values of the counter. An increasing trend on the performance counter might influence the performance architect to consider the possibility of expanding the hardware, or updating the server architecture in the future.

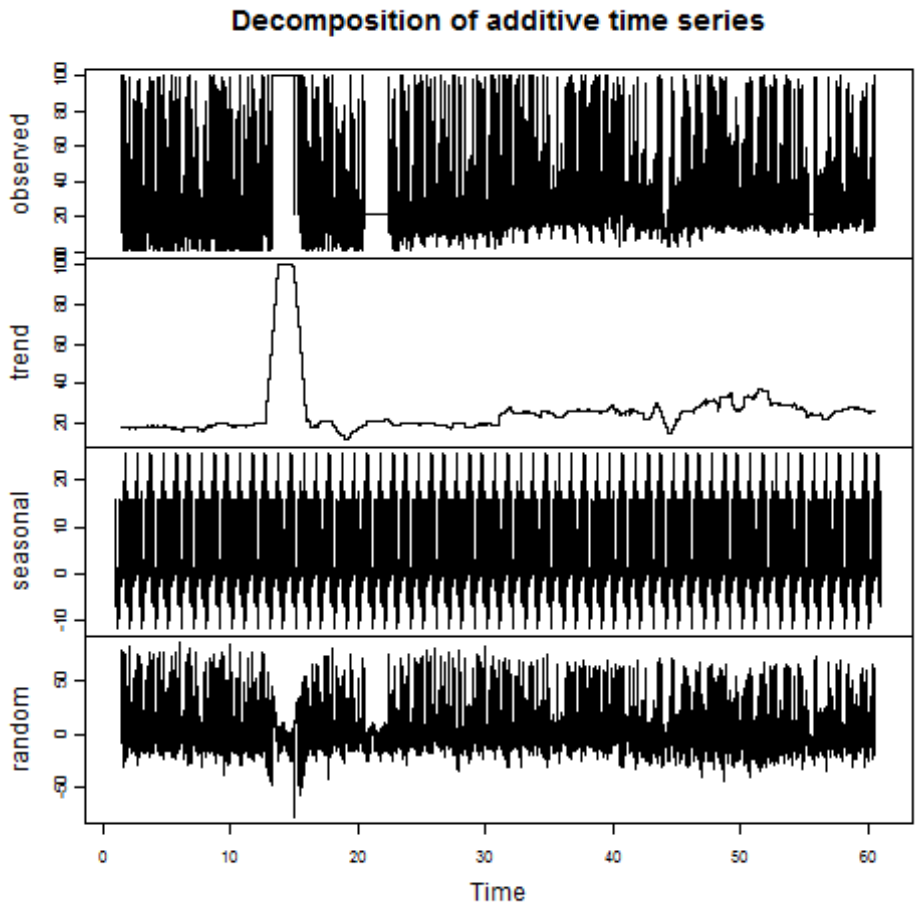


*Figure 2.* Decomposition of the time series for the performance counter %ProcessorTime

**Monitoring Performance of Applications**

As far as monitoring the performance of applications is concerned, we selected which applications were interesting to monitor. That may depend on the type of applications the performance analyst is interested in. For our case, it was interesting to monitor grid and matrix type of applications, which may have several delays when the number of records that have to be loaded or stored is high. We used the related script from our prototype to monitor performance of applications that were used in the test environment of the application from 20th February 2010 until 20th May 2010. We used the prototype to preprocess the dataset (transforming, cleaning, selecting data, etc.) and to prepare the time series object, which goes through decomposition, burst detection and trend estimation. In the end, the knowledge was represented in the format of images, which could be inspected by the performance analysts. An example of the output is provided in figure 3, which illustrates the trend analysis for the duration of BankStatus application (after having removed any identified bursts). As we can roughly see from the fitted line on the graph, the duration of the application has increased for more than 180ms. This is a significant increase that the performance analyst should look further into.

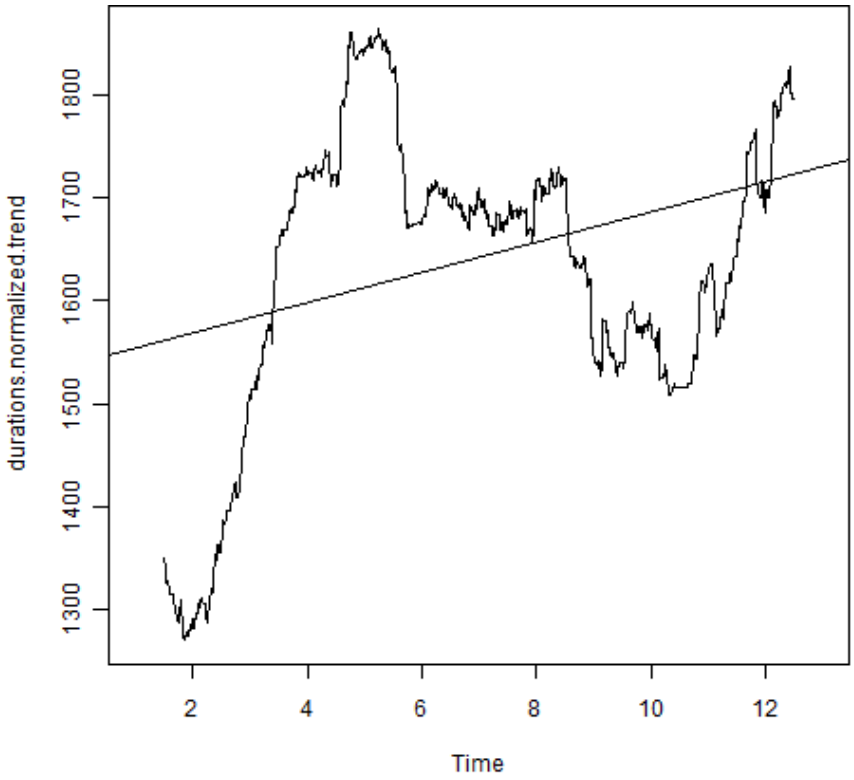


*Figure 3.* Trend estimation for the BankStatus application after removing the burst intervals

**Analyzing Performance of Applications**

In order to analyze the performance of applications, we first had to make a list of the possible performance problems that might arise, similar to the one presented in section 4. Also, since there were many details that influenced the performance of a specific (category of) application(s), we made a list of performance constraints which may influence the duration of applications. We suggested solutions on how to handle these constraints in the analysis, in order to decrease any "noise" in the dataset, which could bias the resulting rules. Considering the hypotheses and the constraints, we made a plan for the association analysis, which included which variables would be included in the logs, what type of response time we would use to find performance problems, etc.

We used the related script from our prototype to analyze the performance of applications from the logs that were collected in three consecutive days, from 6th April 2010 until 8th April 2010. The reason for which we selected these specific days is because several delays in the response time for processing web requests were observed in that time period; hence we would like to find the reasons behind these problems. After preprocessing the data (which contained 441662 rows), we created the binary incidence matrix, which was mined through the Apriori function to find the most frequent itemsets and the rules which had high duration of the right hand side. All the rules that resulted from the association analysis, together with their interestingness measures, were also exported in CSV file format. An excerpt of the most interesting rules with "Duration = high" on the right hand side, is presented in table 2.

Table 2

*Example of Association Rules for High Duration*

| Rule | support | confidence | lift |
|---|---|---|---|
| {HostName=VMWS1, App=FinEntryBankCash.aspx, Action=Save} → {Duration=high} | 0,00687 | 0,61246 | 5,367 |
| {App=FinEntryBankCash.aspx, Action=Save, dayNr=3} → {Duration=high} | 0,00701 | 0,63584 | 5,572 |
| {App=FinEntryBankCash.aspx, Action=Save, PaymentTermCount≥850} → {Duration=high} | 0,00923 | 0,64474 | 5,649 |
| {App=FinEntryBankCash.aspx, Action=Save, GLTransactionsCount≥1000} → {Duration=high} | 0,01292 | 0,60171 | 5,273 |

In this section, we presented how we applied our Performance Mining method and prototype in the case study. We customized our prototype code in order to process the usage logs and provided the performance engineers with all the necessary codes and documentation to perform the same analysis in the future.

## Discussion and Conclusions

In this paper, we tried to answer the research question of how we can detect performance anomalies and their cause in software operation data. Our main contribution is the synthesis of a knowledge discovery method for the domain of software performance evaluation, which has been performed through the pursuit of a meta-algorithmic research approach. Such an approach does not aim for the development of new algorithms, but instead focuses on the extraction of new knowledge through the reuse already proven algorithms in new settings. As Spruit, Vroon, and Batenburg (2014) state, "a meta-algorithmic knowledge discovery approach provides an informatics perspective onto data analytics research by modelling knowledge discovery technology selection to facilitate process analysis to improve people's performance". Thus, we searched for data mining techniques that are particularly applicable for identifying performance problems that appear on the end-user's side; and techniques that could be used to pinpoint the causes of these problems. We suggested a method to apply performance mining in a uniform way. We also validated this method and tested a subset of the proposed techniques through a prototype, which was deployed in the case of an online financial management application.

In the case study that we performed, our method could easily be implemented and was sufficiently apprehended by the stakeholders. Through the activities that were followed, the organization gained insights related not only to the performance of the product, but also, to the way that the performance was analyzed before and how the process could be improved in order to receive more analytical results in an automated way.

The two monitoring tasks gave very satisfactory results: Through the time series decomposition, we managed to extract the trend, seasonal and random movements of the

performance counters and the response times of applications in the long term; the burst detection helped identify burst intervals, which deserve further inspection and analysis from the performance analysts, but also helped remove the noise created in the trend estimation by singular events. The trend analysis helped estimate the trend of the counter/response time, and thus prevent from future performance problems. However, the analysis task, although seemed to be very useful in theory, did not yield satisfactory results. Running the mining for association rules encountered memory allocation problems, due to the static memory allocation in R. Some possible solutions include processing the data in chunks, or adjusting the memory usage settings of R environment, or updating the hardware of the machine on which the analysis takes place. Further research should be conducted in the performance analysis task. An implementation of association rules mining in another language, an update of the system hardware, or the experimentation with other data mining techniques (e.g. exceptional model mining) should be considered. Nevertheless, even the current state of the prototype, at least provides an automated technique to associate performance problems with possible reasons, and can at least find the evident ones.

In the future, it would be interesting to extend the method and prototype to analyze software operation data of more products, but also to try out other data mining techniques that were suggested for the related tasks. From a technical perspective, this research could also be extended to the performance analysis of very large software products, such as Twitter or Facebook; in such large settings we may need to explore the combination of data mining techniques with techniques on distributed computing on large data sets, on clusters of computers (e.g. MapReduce). From a business perspective, it would be interesting to study how the performance mining method could help automate the performance evaluation of software products, and whether it would succeed in reducing costs and increasing customer satisfaction (McKinney et al., 2002).

In conclusion, research contributes to the domain of knowledge management in software organizations (Bjørnson & Dingsøyr, 2008; Lindvall & Rus, 2002), and specifically to the extraction of knowledge on how software operates in the field (van der Schuur et al., 2010). In addition, this research contributes to the Software Performance Evaluation domain, by suggesting specific data mining techniques, and a method to employ them, in order to performance anomalies and their causes from software operation data. We expect that the performance knowledge that can be derived from software operation data may be used to support the processes of software product management, development and maintenance (Midha & Bhattacherjee, 2012).

## References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proc. 20th International Conference on Very Large Data Bases*, San Francisco, CA, USA.

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *ACM SIGMOD Record, 22*(2), 207-216.

Ammons, G., Ball, T., & Larus, J. R. (1997). Exploiting hardware performance counters with flow and context sensitive profiling. *ACM SIGPLAN Notices*, *32(5)*, 85-96.

Andrews, J. H., & Zhang, Y. (2003). General test result checking with log file analysis. *Software Engineering, IEEE Transactions on*, *29*(7), 634-648.

Avritzer, A., Kondek, J., Liu, D., & Weyuker, E. J. (2002). Software performance testing based on workload characterization. In *Proceedings of the 3rd International Workshop on Software and Performance* (pp. 17-24). ACM.

Bjørnson, F. O., & Dingsøyr, T. (2008). Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, *50*(11), 1055-1068.

Bolloju, N., Khalifa, M., & Turban, E. (2002). Integrating knowledge management into enterprise environments for the next generation decision support. *Decision Support Systems*, *33*(2), 163-176.

Borzemski, L. (2009). Towards Web performance mining. In *Web Mining Applications in E-commerce and E-services* (pp. 81-102). Springer Berlin Heidelberg.

Cowpertwait, P. S., & Metcalfe, A. V. (2009). *Introductory time series with R* (pp. 51-55). New York: Springer.

Courtney, J. F. (2001). Decision making and knowledge management in inquiring organizations: toward a new decision-making paradigm for DSS. *Decision Support Systems*, *31*(1), 17-38.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). CRISP-DM 1.0 Step-by-step data mining guide.

Delen, D., & Demirkan, H. (2013). Data, information and analytics as services. *Decision Support Systems*, *55*(1), 359-363.

Demirkan, H., & Delen, D. (2013). Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud. *Decision Support Systems*, *55*(1), 412-421.

Gunter, D., & Tierney, B. (2003). NetLogger: A toolkit for distributed system performance tuning and debugging. In *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on* (pp. 97-100). IEEE.

Hahsler, M., Grün, B., Hornik, K., & Buchta, C. (2009). Introduction to arules–A computational environment for mining association rules and frequent item sets. *The Comprehensive R Archive Network*.

Hahsler, M., Grün, B., & Hornik, K. (2005). Arules a computational environment for mining association rules and frequent item sets, *Journal of Statistical Software, 14*, 1–25.

Han, J. (2005). *Data Mining: Concepts and Techniques*, San Francisco, CA: Morgan Kaufmann Publishers Inc.

Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, *15*(1), 55-86.

Heinrichs, J. H., & Lim, J. S. (2003). Integrating web-based data mining tools with business models for knowledge management. *Decision Support Systems*, *35*(1), 103-112.

Hevner, A., & Chatterjee, S. (2010). *Design science research in information systems* (pp. 9-22). Springer US.

Hevner, A. R., March, S.T., Park, J., & Ram, S. (2002). Design Science in Information Systems Research. *MIS Quarterly 28*(1), 75-105.

Imberman, S. P. (2001). Effective use of the KDD process and data mining for computer performance professionals. In *Proceedings of International Computer Measurement Group Conference* (pp. 611-620).

Jain, R. (2008). *The art of computer systems performance analysis*. John Wiley & Sons.

Johnson, M. J., Maximilien, E. M., Ho, C. W., & Williams, L. (2007). Incorporating performance testing in test-driven development. *Software, IEEE*, *24*(3), 67-73.

Lindvall, M., & Rus, I. (2002). Knowledge management in software engineering. *IEEE software*, *19*(3), 0026-38.

Maimon, O., & Rokach, L. (Eds.). (2005). *Data mining and knowledge discovery handbook* (Vol. 2). New York, NY: Springer.

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., & Ghalsasi, A. (2011). Cloud computing-The business perspective. *Decision Support Systems*, *51*(1), 176-189.

McKinney, V., Yoon, K., & Zahedi, F. M. (2002). The measurement of web-customer satisfaction: An expectation and disconfirmation approach. *Information Systems Research*, *13*(3), 296-315.

Midha, V., & Bhattacherjee, A. (2012). Governance practices and software maintenance: A study of open source projects. *Decision Support Systems*, *54*(1), 23-32.

Moreno García, M. N., Quintales, L. A. M., García Peñalvo, F. J., & Polo Martín, M. J. (2004). Building knowledge discovery-driven models for decision support in project management. *Decision Support Systems*, *38*(2), 305-317.

Musumeci, G. P., & Loukides, M. K. (2002). *System performance tuning*. Sebastopol, CA: O'Reilly Media, Inc.

Pachidi, S., Spruit, M., & Weerd, I. van de (2014). Understanding Users' Behavior with Software Operation Data Mining. *Computers in Human Behavior, 30*, 583–594.

Pray, K. A., & Ruiz, C. (2005). Mining expressive temporal associations from complex data. In *Machine Learning and Data Mining in Pattern Recognition* (pp. 384-394). Springer Berlin Heidelberg.

Spruit, M., Vroon, R., & Batenburg, R. (2014). Towards healthcare business intelligence in long-term care: An explorative case study in the Netherlands. *Computers in Human Behavior, 30*, 698-707.

Sun, Y., He, S., & Leu, J. Y. (2007). Syndicating Web Services: A QoS and user-driven approach. *Decision Support Systems*, *43*(1), 243-255.

Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining*. Boston, MA: Addison Wesley Longman Publishing Co., Inc.

Team A. C. E. (2003). *Performance Testing Microsoft .NET Web Applications*. Redmond, WA: Microsoft Press.

Team R. C. (2012). *R: A language and environment for statistical computing*.

Thakkar, D., Hassan, A. E., Hamann, G., & Flora, P. (2008). A framework for measurement based performance modeling. In *Proceedings of the 7th International Workshop on Software and Performance* (pp. 55-66). ACM.

van der Schuur, H., Jansen, S., & Brinkkemper, S. (2010). A reference framework for utilization of software operation knowledge. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on* (pp. 245-254). IEEE.

Venables, W. N., Smith, D. M., & R Core Team (2014). *An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics.* Retrieved September 9, 2014, from http://cran.r-project.org/doc/manuals/r-patched/R-intro.pdf

Vilalta, R., Apte, C. V., Hellerstein, J. L., Ma, S., & Weiss, S. M. (2002). Predictive algorithms in the management of computer systems. *IBM Systems Journal*, *41*(3), 461-474.

Vlachos, M., Wu, K. L., Chen, S. K., & Philip, S. Y. (2008). Correlating burst events on streaming stock market data. *Data Mining and Knowledge Discovery*, *16*(1), 109-133.

Vleugel, A., Spruit, M., & Daal, A. van (2010). Historical data analysis through data mining from an outsourcing perspective: the three-phases method. *International Journal of Business Intelligence Research, 1*(3), 42–65.

Warren, C. E., Payne, D. V., Adler, D., Stachowiak, M., Serlet, B. P., & Wolf, C. A. (2010). *U.S. Patent No. 7,644,397*. Washington, DC: U.S. Patent and Trademark Office.

van de Weerd, I., & Brinkkemper, S. (2009). Meta-Modeling for Situational Analysis and Design Methods. In M. Syed, & S. Syed (Eds.) *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications* (pp. 35-54). Hershey, PA: Information Science Reference.

Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M.I. (2009). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating Systems Principles* (pp. 117-132). ACM.

Zhu, J. (2014). *Quantitative models for performance evaluation and benchmarking: data envelopment analysis with spreadsheets* (Vol. 213). Springer.

Zo, H., Nazareth, D. L., & Jain, H. K. (2010). Security and performance in service-oriented applications: Trading off competing objectives. *Decision Support Systems*, *50*(1), 336-346.