# Automatic, Load-Independent Detection of Performance Regressions by Transaction Profiles

Shadi Ghaith, Miao Wang, Philip Perry, and John Murphy
School of Computer Science and Informatics
University College Dublin, Ireland
shadi.ghaith@ucdconnect.ie

## ABSTRACT

Performance regression testing is an important step in the production process of enterprise applications. Yet, analysing this type of testing data is mainly conducted manually and depends on the load applied during the test. To ease such a manual task we present an automated, load-independent technique to detect performance regression anomalies based on the analysis of performance testing data using a concept known as Transaction Profile. The approach can be automated and it utilises data already available to the performance testing along with the queueing network model of the testing system.

The presented "Transaction Profile Run Report" was able to automatically catch performance regression anomalies caused by software changes and isolate them from those caused by load variations with a precision of 80% in a case study conducted against an open source application. Hence, by deploying our system, the testing teams are able to detect performance regression anomalies by avoiding the manual approach and eliminating the need to do extra runs with varying load.

## Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering

## General Terms

Measurement, Performance

## Keywords

Application change, performance models, regression testing

## 1. INTRODUCTION

Performance regression testing is an important step of software production process and has the goal of ensuring that a newer version of the software performs no worse than the previous versions of the same software [1]. To achieve this, the application is exposed to a field-like load via load generator software such as HP LoadRunner [2] and the open source tool JMeter [3]. This load is applied for an extended period of time and as a result a huge amount of performance counters will be collected and analysed for performance regression anomalies.

Virtual users interact with the enterprise application under test by triggering various transactions such as logging in to the system or checking out an item. These transactions propagate through the application tiers to fulfil the users request. The time required to complete each transaction is an important metric to assess the application's quality.

During the performance run the following two types of data are collected and analysed.

1. Transaction Response Time (TRT): Measures the time required to process a request by all system resources [4]. It is generated by the load generator software along with the transaction types and rates.

2. Resources Utilization (RU): Represents the utilization of various system resources (CPU, Disk I/O and Network). It is produced by the system monitoring tools such as NMON, Perfmon and TCPDump on each server.

The test engineer will then investigate those counters to look for performance regression anomalies, which include any increase of TRT or RU [5]. For example if the TRT of a certain transaction is increased from 0.32 sec to 0.39 sec then the test run will be marked as a Fail. Otherwise the run is declared as Pass. This process is depicted in Figure 1.

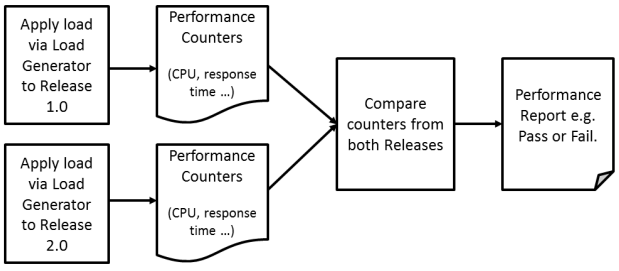


**Figure 1: Performance Regression Testing Process [5].**

In addition to comparing new and previous releases of the software, it is also important to compare the performance



*JAMAICA'13*, July 15, 2013, Lugano, Switzerland
Copyright 2013 ACM 978-1-4503-2161-7/13/07...$15.00
http://dx.doi.org/10.1145/2489280.2489286

between software runs within the development process. This is important because it is only at this early stage such regressions can be actively analysed and fixed.

In the process of detecting performance regression anomalies, the testing team faces the following two challenges:

1. Manual Process: The manual inspection of performance counters in search of anomalies is error-prone and time consuming. Furthermore, the amount of data collected from performance test runs is large and may contain hundreds of performance counters. In addition, given that the performance regression testing process is usually held late in the project lifecycle [1], the time required for this manual analysis is often not available, in which case an automated approach is required [6].

2. Load Variations: Frequently, the new performance test runs are conducted with a different load than the previous release to account for varying field-like scenarios [7]. Hence, an increase of any performance counter (i.e. TRT or RU) could be caused by either the increase of the load applied to the system or an anomaly in the new software. To distinguish between these two cases, an extra performance run with a load similar to the previous release would be needed which is time and resource consuming. Hence, a new performance regression anomaly detection approach should not be sensitive to the load applied to the system.

In this paper, we combine concepts from queueing network representation of computer systems as used in the capacity management process, with the performance regression testing concepts to achieve an automated, load independent approach to detect performance regression anomalies caused by a software update. Our approach should speed up the lengthy performance regression testing process and consquently speed up the overall software production process and consequently result in reduced development cost and time to the market.

To achieve this, we present the use of the output of the performance regression testing run (TRT and RU) along with the Queueing Network Model (QNM) of the system under test, such as the one shown in Figure 4, to calculate the so called Transaction Profile [8] which we believe reveals important characteristics of the transactions performance. By this we extend on our previous work [9] where we suggested measuring the TP using a load generator software.

# 2. PRESENTED TRANSACTION PROFILE (TP) APPROACH

## 2.1 What Is the TP?

To fulfil users' requests, a software transaction propagates through the various system resources, such as CPU and Disk I/O, across the various servers in all the system tiers [8]. The total time required to process the requests on all resources is the TRT. It includes the time spent on each resource unit as well as in its queue.

It is worth mentioning that the request may visit each resource multiple times and requires a certain service time at each visit. At the same time, the request may wait in resource queues if the processing unit of the corresponding resource is in use by another request. Additionally, the Service Demand is defined as the total time required to process a request on a certain resource during all visits, excluding the time spent in the queue [10] [11].

Given the above, the TP is defined as the series of service demands of a request (transaction) over all system resources [8]. In another words, the TP is the TRT when the request is the only one in the system (no queues), or the TP is the minimum bound of the TRT. The TP is usually represented as a horizontal bar as shown in Figure 2.

From a closer look at the TP, we can see that it depends on the service time on each resource and the number of visits to that resource which can only be determined by the software characteristics (implementation). For example, the TP may change if the software is modified to require an extra visit to the hard disk. At the same time the TP is not dependent on the load applied to the system as, by definition, it does not include the time spent in the queue. Hence, the TP represents the fundamental characteristics of the software and is independent of the load.

The TP is not a metric that corresponds to a real usage scenario, instead it is just a hypothetical concept used with QNM of software systems to predict system performance as is shown in Section 3.

## 2.2 Presented TP Usage in Regression Testing

Given the above description, we present the use of the TP to detect changes of the software that affect its performance.

Our hypothesis is:

*An automated (or visual) comparison of the TPs between two releases can highlight performance regression anomalies caused by software updates as opposed to those caused by load variations.*

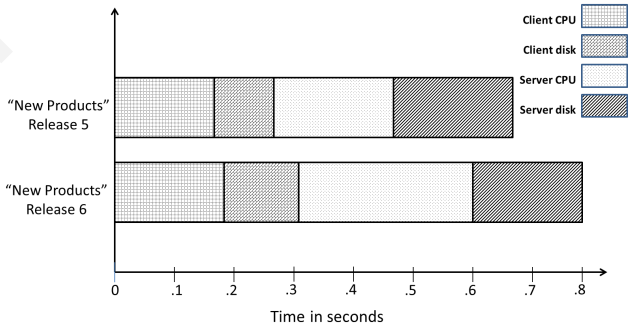


**Figure 2: Comparison of the TP for the "New Products" Transaction Over Two Releases.**

As shown in Figure 2, the TPs of the "New Products" transaction over two releases show an increase of the TP in the later release. This is considered a performance regression anomaly per our hypothesis. A closer look at Figure 2 shows that the Server CPU component of the TP contributes most to the increase of the TP.

The use of TP as described above fixes the two issues of performance regression testing, the manual approach and the load dependency.

For this paper, we assume that the hardware platform is fixed between the two releases. The topic of hardware evolution will be explored in our future work.

## 2.3 TP Run Report

To promote the use of our presented TP approach we de-

signed the TP Run Report shown in Figure 3. The TP Run Report is produced by applying our presented technique against the normal performance run data and the TPs from the previous run. The TP Run Report consists of the TP Report Summary and the Detailed TP Graph.

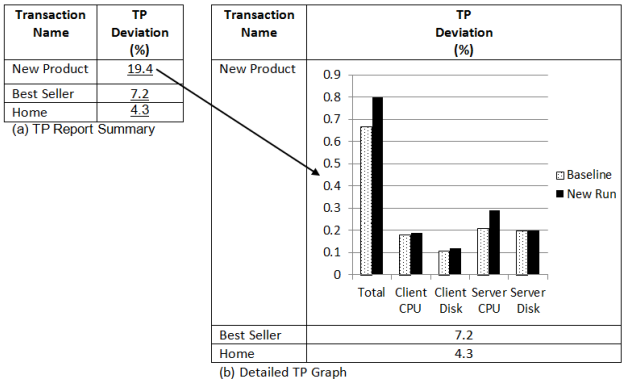


**Figure 3: TP Run Report.**

### 2.3.1 *TP Report Summary*

The TP Report Summary shown in Figure 3a provides a list of transactions for which the TP in the new release deviates (increases) from its value in the previous release by a value above a pre-determined threshold (for example 3%). Those are the transactions which are declared by our approach as potential performance regression anomalies.

### 2.3.2 *Detailed TP Graph*

To get an insight on each of those anomalies, the testing engineer may click on the link under each TP deviation value to show the corresponding Detailed TP Graph which is shown in Figure 3b. Such a graph provides information about actual values of the TPs as well as the hardware resources making up the TP and the contribution of each of them to the detected deviation. In this example we can see that the Server CPU contributes most to the anomaly.

# 3. QUEUEING NETWORKS OF COMPUTER SYSTEMS

## 3.1 Queueing Network Model

Computer Systems can be represented by a QNM [8] [11] [12] as the one presented in Figure 4. It consists of multiple nodes, where each node represents a hardware resource such as CPU, Disk I/O and Network. Users initiate request at the Users node that will pass through the various nodes to fulfil the user request. Each node consists of a processing unit and a queue. The processing unit will serve the request if it is free or the request will have to wait in the queue [11]. A request may visit each resource one or more times.

In the queueing network terminology, the nodes are called stations, the transaction types are classes, and the transaction is called a customer [12]. The time required to serve a customer by each node on one visit is known as the service time. The time required to service the request on a certain node during all visits is known as service demands [12] (the product of the service time and number of visits).

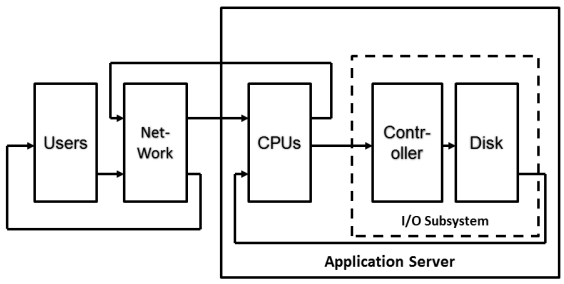


**Figure 4: QNM of a Computer System.**

Given the stations, classes, customers incoming rate, the service times and number of visits, QNM such as the one shown in Figure 4 can be solved [13] to get the response time (TRT) and utilization (RU) of the various classes and stations respectively.

## 3.2 Solving Queueing Network by Service Demands

To solve a QNM one needs to specify the service times and number of visits to resources along with the other parameters. This allows to get the various system level counters, such as system TRT and throughput, and the per-station counters, such as the RU, throughput and residence times (total time spent on each station). But if the system satisfies the BCMP hypothesis [13] [14], then the service demand may be used instead of the service time and number of visits [15] in which case the QNM can be represented as shown in Figure 5 [15]. In this case, we cannot obtain some of the per-station level counters like the stations throughputs and residence times, since this is a level of detail that we may model only if we use the service times and visits.

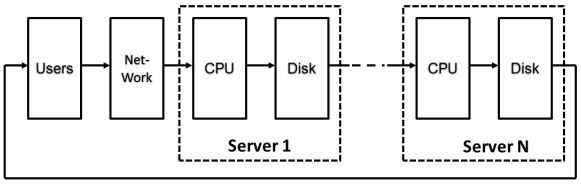


**Figure 5: Representation of Computer System using Service Demands Instead of Service Times and Visits to Resources.**

Given that we deal with service demands, and not the service times and number of visits, and adding to this that the computer resources approximately satisfy [16] the BCMP hypothesis, we conclude that we can represent computer systems as shown in Figure 5 when we solve the QNM [17].

# 4. PRESENTED APPROACH DETAILS

## 4.1 Normal Solving of Queueing Networks

To solve a QNM such as the one shown in Figure 5, the following three aspects are needed.

1. QNM of the system.

2. Load characterization, mainly the transactions types (classes) and rates.

3. Service demands (and so the TP) of the various transactions (classes) on each resource (station).

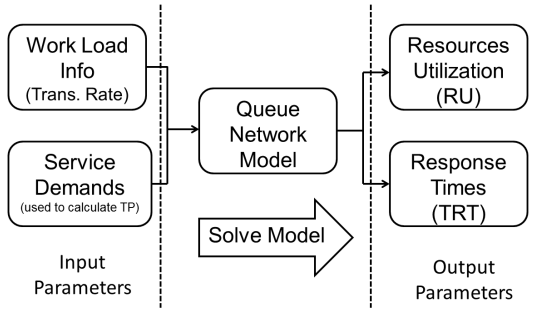


**Figure 6: Normal Solving of Queueing Networks.**

Knowing the above, the model can be solved via tools such as the Java Modelling Tool (JMT) [18] to get useful information about the system, mainly the TRT and RU for the various transactions (classes) and resources (stations) respectively [13]. This process is depicted in Figure 6. This process is used in various fields, for example the capacity management of computer systems.

In performance regression testing, we have the two outputs in Figure 6 which are the TRT and RU. The Load info and the QNM can be found from the testing system and load generator software. Yet, the service demands are the target of the performance regression testing (to calculate the TP) and so we present an inversed process as shown in Figure 7.

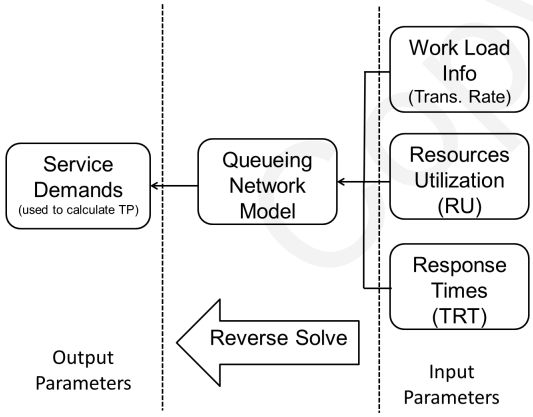


**Figure 7: Presented Approach to Obtain TP from Performance Testing Data.**

## 4.2 Reverse Solve Queueing Networks

We present a search based approach to reverse solve the QNM to obtain the service demands, and accordingly the TPs, as shown in Figure 8.

An initial TP value is used to solve the model (the load is fixed across the process and so it is omitted from Figure 8). The initial TP value can be the TP from the previous run or

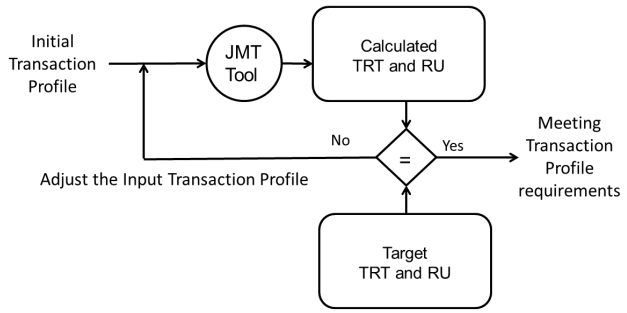


**Figure 8: Reverse solve Queueing Network via JMT Using a Search Based Approach.**

it can be measured via a single user test. It can also be set to a random value, but this will increase the time required to converge the search.

Furthermore, the model is solved via JMT and the TRT and RU are calculated and subsequently compared to the target TRT and RU (which are the outputs of performance testing). If those mismatched, a new value of the TP will be nominated based on the difference between the calculated and target TRT and RU. This will keep going until the calculated and target TRT and RU match, in which case the target TP is found and can be used to compare to TP from previous run as shown in the next section. The search stops when the required TP is found or when a loop is detected, which could happen when the QNM does not accurately reflect the system. There are two reasons for QNM inaccuracies. First, due to a missing resources caused by dropping the performance counters from one of the servers. Second, QNM can be inaccurate if the software resources, such as the number of threads and available database data sources, form the real bottleneck of the system.

## 4.3 Overall Presented Outline

Figure 9 shows the diagram of the overall presented process. The New Run Data is used as an input to this process. The load generator files are parsed to obtain transactions information which includes the TRT, transaction types and rates. The system monitoring files are parsed to get the RU and resources types. The QNM, such as the one shown in Figure 5, is then built by mapping the transaction types to classes and resources types to stations. At the next step, all of the QNM, TRT, RU and transaction rates are used to reverse solve the system to get the TP for all transactions as shown in Figure 7. At the final step, those TPs are compared to the TP Previous Run(s) to generate the TP Run Report similar to the one shown in Figure 3.

# 5. CASE STUDY

## 5.1 System Information

We conducted our tests and an open source web 2.0 application JPetStore 6.0 [19]. It is built on JEE technology and simulates an application to offer various types of pets online. The application runs on a Tomcat Application Server and was modified to run on a MySQL Database Server. The tests were performed on the set of transactions shown in Transaction column in Table 1.

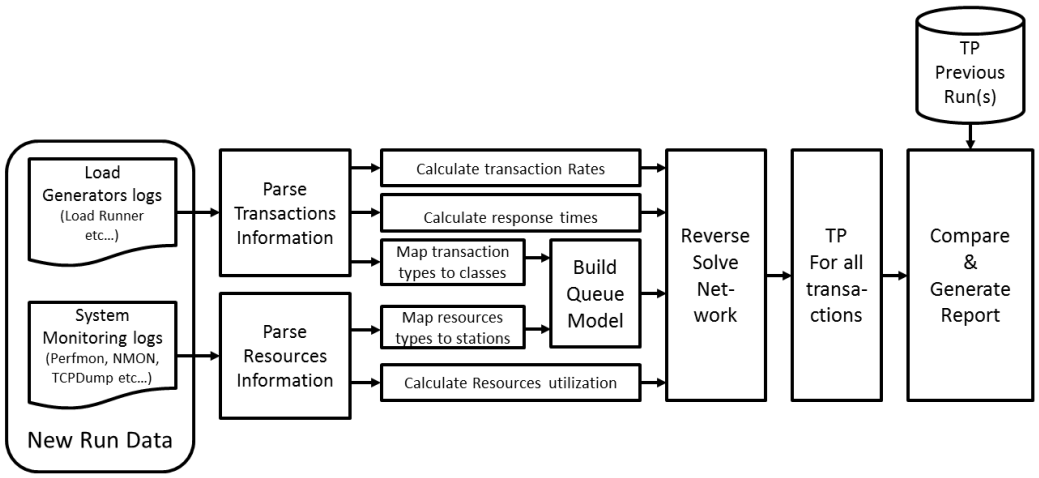


**Figure 9: Our Presented Process.**

**Table 1: Defects Injected "Individually" in Various Transactions and the Resulted TP Deviation.**

| Run | Transaction | Injected Anomaly | TP Deviation (from baseline) |
|---|---|---|---|
| 1 | Home. | No index on corresponding table's key columns. | 20% |
| 2 | List Products. | | 7% |
| 3 | List Items. | | 5% |
| 4 | Checkout. | | 4% |
| | | | |
| 5 | List Items. | Enabling logging on transaction code. | 5% |
| 6 | Add to Cart. | | Not caught |
| 7 | Checkout. | | Not caught |
| | | | |
| 8 | Home. | Not limiting the database query. | 7% |
| 9 | List Products. | | 6% |
| 10 | List Items. | | 4% |

The system was deployed as 3 tiers architecture. The client machine runs the JMeter load generator software. The Application Server contains the Tomcat Application Server on which the JPetStore 6.0 application is deployed. While on the Database Server the MySQL server is deployed and contains the application database which was populated by an SQL script that we generated.

## 5.2 Case Study Design

First a baseline run was executed with the original application and the TPs were found and recorded. Then a total of 10 runs were executed on modified versions of the application. In each of these versions one transaction code was modified to contain a performance defect from those commonly introduced by developers. The TP Run Report was generated for those builds and investigated to find if those defects were really caught. Also, the Detailed TP Graph of each anomaly was investigated to see if it relates to the nature of the bug introduced. Additionally, we made sure no TP deviation was noticed on the transactions for which the code has not changed.

In our study we define precision as:

$$Precision = (1 - \frac{N}{T}) * 100\%$$

N: Number of tests, each with one injected defect that has not been caught.

T: Total number of tests, each with one injected defect.

We used a set of the common performance defects as shown in the Injected Anomaly column in Table 1.

Although we introduced one performance defect at a time, the technique works well with multiple defects. This will be explored more in our future work when we study a real Enterprise Application. Generating one TP Run Report took less than a minute for this simple topology.

## 5.3 Results

The tests were performed with JMeter to simulate 35 users running the transactions in Table 1 by a certain sequence. After generating the TP Run Report of the various runs, we found that 8 of the 10 injected anomalies were caught, indicating a precision of 80%.

In the first 4 runs of Table 1, we got the four cases caught with considerable deviation of 20% for the Home transaction and for the List Products, List Items and Checkout Items we recorded 7%, 5% and 4% respectively. Investigating the Detailed TP Graph we found that the Database Server CPU was the main contributor to the TP deviation which is expected from this kind of defects. But a little change was recorded from the hard disk of the Database Server which could be caused by the fact that the system has a huge RAM and so most of the database is already loaded into it.

In the following 3 runs, we got only one defect caught by the TP Run Report which is for the List Items transaction and with a deviation of 5%. This can be explained by the fact that the amount of logging statements added to this small application will not have a noticeable impact on such powerful machines.

In the last 3 runs, all defects have been caught with a deviation of 7%, 6% and 4% respectively. Looking into the Detailed TP Graph we found that the major deviation was

caused by CPU of the Application Server as expected. But we did not find any contribution of the network as we expected but this could be due to the relatively low amount of data in the database and the fast network connection.

# 6. RELATED WORK

The manual approach is the most common way used in industry to analyse performance runs data looking for regression anomalies. The testing engineer will have to manually analyse performance data looking for an uptrend in RU of one of the server's resources or an increase in the TRT of a transaction.

Statistical techniques were among the first approaches to be proposed by researchers to automate the detection of performance regression anomalies [7] [20]. They proposed the use of Statistical Process Control charts (SPC) which is used to control the quality of various processes in industry. The old runs data (baseline) is used to draw the Upper Control Limit (UCL) and Lower Control Limit (LCL) as well as the Centre Line (CL). Then the new runs data is tested against those limits and a violation ratio is calculated to represent the up normal values. If this violation ratio exceeds a certain pre-determined threshold an anomaly is raised. Yet, this approach is still sensitive against load variations as it will not distinguish between changes due to load changes from those due to software updates.

Inferring the service demands (and so the TP) from TRT and RU has been proposed before by some studies. This is done because measuring TRT and RU is much easier than measuring the service demands. Casale et al [21] suggested to calculate the service demand from the RU by solving a linear regression between them. They assume multiple runs with various loads are provided. This approach assumes the system can be modelled by a single server and that multiple runs with different loads can be provided. The complexity of the systems under test and the resources and time constraint make those assumptions inadequate to be used in performance regression testing.

# 7. CONCLUSIONS AND FUTURE WORK

In our research, we showed that the TP can be automatically generated and provide a simple graphical aid to analyse performance testing data to detect performance regression anomalies. Furthermore, the TP is independent of the load applied to the system which saves time and resources running extra performance runs with a varying load.

The presented TP Run Report approach was verified in an open source web application and proved to be able to detect performance regression anomalies with a precision of 80%.

Our automated, load-independent approach improves the performance testing process, reduces the cost and time and therefore improves the software production process.

It is a major advantage to be able to build the QNM automatically in order to automate the entire process and save the testing team from the burden of maintaining the QNM of the system. Yet the accuracy of the BCMP approximation, used to facilitate the automated building of the QNM, is another area that requires more research to find any constraints to its applicablity to various types of web 2.0 traffic.

The topic of hardware evolution between releases will also be explored in our future work.

# 8. ACKNOWLEDGMENTS

Supported, in part, by Science Foundation Ireland grant 10/CE/I1855.

# 9. REFERENCES

[1] J. Z. Ming. Automated analysis of load testing results. In *Proceedings of the 19th international symposium on Software testing and analysis*, New York, NY, USA, 2010. ACM.

[2] Hewlett-Packard Development Company. Loadrunner. http://www8.hp.com/ie/en/software-solutions/software.html?compURI=1175451#.UQ-2QWfuo-A, 2013.

[3] The Apache Software Foundation. Apache jmeter. http://jmeter.apache.org/, 2013.

[4] R. Jain. *The art of computer systems performance analysis*, volume 182. John Wiley & Sons New York, 1991.

[5] Z. M. Jiang A. Hassan M. Nasser T. H.D. Nguyen, B. Adams and P. Flora. Automated detection of performance regressions using statistical process control techniques. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, New York, NY, USA, 2012.

[6] T. Gao, Y. Ge, and J. Ni G. Wu. A reactivity-based framework of automated performance testing for web applications. In *Proceedings of DCABES*, pages 593–597. IEEE, 2010.

[7] F. M. Bereznay. Did something change? using statistical techniques to interpret service and resource metrics. In *32nd International Computer Measurement Group Conference, December 3-6, 2006, Reno, Nevada, USA, Proceedings*. Computer Measurement Group, 2006.

[8] L. Grinshpan. *Solving Enterprise Applications Performance Puzzles*, pages 5–57. John Wiley and Sons, Inc., Hoboken, New Jersey, 2012.

[9] S. Ghaith, M. Wang, P. Perry, and J. Murphy. Profile-based, load-independent anomaly detection and analysis in performance regression testing of software systems. In *17th European Conference on Software Maintenance and Reengineering (CSMR'13)*, Genova, Italy, 2013.

[10] D.A. Menascé and V.A.F. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall, 2002.

[11] J. Walrand. *An introduction to queueing networks*, volume 165. Prentice Hall Englewood Cliffs, NJ, 1988.

[12] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *Software Engineering, IEEE Transactions on*, 35(2):148–161, 2009.

[13] R.O. Baldwin, N.J. Davis Iv, S.F. Midkiff, and J.E. Kobza. Queueing network analysis: concepts, terminology, and methods. *Journal of Systems and Software*, 66(2):99–117, 2003.

[14] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM, 22(2)*, pages 248–260, 1975.

[15] G. Serazzi. Performance evaluation modelling with jmt: learning by examples. technical report 366, Politecnico di Milano - DEI, Milano, 2008.

[16] D.A. Menasce, L.W. Dowdy, and V.A.F. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[17] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. *Quantitative System Performance*. Prentice-Hall, 1984.

[18] M.Bertoli, G.Casale, and G.Serazzi. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In *Proceedings of QEST 2006 Conference*. IEEE Press, Sep 2006.

[19] MyBatis. Jpetstore 6. http://www.mybatis.org/spring/sample.html/, 2013.

[20] T H. D. Nguyen. Using control charts for detecting and understanding performance regressions in large software. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, Washington, DC, USA, 2012. IEEE Computer Society.

[21] G. Casale, P. Cremonesi, and R. Turrin. Robust workload estimation in queueing network performance models. In *Proceedings of PDP*, Washington, DC, USA, 2008. IEEE Computer Society.