A Framework to Evaluate the Effectiveness of Different Load Testing Analysis Techniques

Ruoyu Gao[1], Zhen Ming (Jack) Jiang[2], Cornel Barna[3] and Marin Litoiu[4]
Department of Electrical Engineering and Computer Science[1,2,3], School of Information Technology[4]
York University, Toronto, Canada[1,2,3,4]
{rgao[1], zmjiang[2], cornel[3]}@cse.yorku.ca, mlitoiu@yorku.ca[4]

***Abstract*—Large-scale software systems like Amazon and eBay must be load tested to ensure they can handle hundreds and millions of current requests in the field. Load testing usually lasts for a few hours or even days and generates large volumes of system behavior data (execution logs and counters). This data must be properly analyzed to check whether there are any performance problems in a load test. However, the sheer size of the data prevents effective manual analysis. In addition, unlike functional tests, there is usually no test oracle associated with a load test. To cope with these challenges, there have been many analysis techniques proposed to automatically detect problems in a load test by comparing the behavior of the current test against previous test(s). Unfortunately, none of these techniques compare their performance against each other.**

In this paper, we have proposed a framework, which evaluates and compares the effectiveness of different test analysis techniques. We have evaluated a total of 23 test analysis techniques using load testing data from three open source systems. Based on our experiments, we have found that all the test analysis techniques can effectively build performance models using data from both buggy or non-buggy tests and flag the performance deviations between them. It is more cost-effective to compare the current test against two recent previous test(s), while using testing data collected under longer sampling intervals ($\geq$ 180 seconds). Among all the test analysis techniques, Control Chart, Descriptive Statistics and Regression Tree yield the best performance. Our evaluation framework and findings can be very useful for load testing practitioners and researchers. To encourage further research on this topic, we have made our testing data publicity available to download.

I. Introduction

Many large-scale software systems (e.g., BlackBerry's telecommunication systems [1] and Microsoft's web services [2]) are used by thousands and millions of users everyday. Study finds that most failures in the field are due to systems not able to scale rather than feature bugs [3]. The failure to scale could result in catastrophic consequences and unfavorable media coverage (e.g., the botched launch of Apple's MobileMe [4] and US Government's Health Care Website [5]). Hence, to ensure the quality of the system in the field, load testing is a required testing procedure in addition to the conventional functional testing (e.g., unit and system integration testing).

Load testing is usually conducted after all the functional testing is completed [6]. It uses one or more load generators, which mimic hundreds or thousands of real users accessing the system at the same time (called the *load*). During the course of the load testing, there are two types of system behavior data generated: execution logs and counters. Execution logs are generated by debug statements (e.g., printf and System.out.println) that developers insert into the code. Counters record the workload and the resource usage (e.g., CPU, memory and disk utilization) of system during runtime. Since the goal of a load test is to assess the system behavior under load, the duration of a load test can be a few hours (e.g., 8 hours for a working day) or even weeks. Hence, the volume of the generated system behavior data can be huge. Unfortunately, analyzing the results of a load test is challenging due to the following three reasons:

1) **Lack of Test Oracle:** Unlike functional testing, which has clear pass and fail criteria, the objectives of a load test are not clear [7] and are often defined later on a case-by-case basis [8].
2) **Large Volume of Data:** A load test can generate hundred megabytes or even terabytes of system behavior data. It would be impossible for load testing practitioners to analyze them manually.
3) **Limited Time:** Load testing is usually the last step in the already delayed software development cycle. The time allocated for test analysis can be very limited.

To cope with the above challenges, there have been a few techniques proposed to automatically analyze the results of a load test (e.g., [1], [9], [10], [11]). These techniques automatically derive the performance models from the previous test(s) using various approaches (e.g., queuing theory or data mining) and check whether the behavior of the current test matches with the previous test(s). However, none of the aforementioned works compared the effectiveness of their techniques against others nor provided replication packages. We suspect this is mainly due to two reasons: (1) most of the studies are conducted on commercial systems, whose test results cannot be made available to the public due to confidentiality concerns; and (2) certain details are missing. For example, many tools described in these works are not publicity available to download. In addition, many techniques impose threshold values. For example, the CPU utilizations in the current test maybe anomalous, as they are 30% above the predicted values. However, the exact threshold values are either not disclosed or should be tunable based on different systems. In this paper, we have re-implemented and systematically evaluated the effectiveness of 23 different test analysis techniques. The contributions of this paper are:

1) This is the first study, which systematically evaluates the effectiveness of different test analysis techniques used in a load test.
2) Our evaluation framework is very flexible. It evaluates different test analysis techniques on a range of threshold values. In addition, it can easily integrate new test analysis techniques or new test data.
3) The test data used in this paper is made available online [12] to encourage further research on this topic. Our data is a collection of system behavior data generated by load tests executed on three open source systems. For each system, load testing is conducted both on "buggy" versions, which contain manually injected or real-world performance bugs, and on "healthy" versions. Such data takes a long time to prepare and is rarely available.

Paper Organization

The rest of the paper is organized as follows: section II provides some background knowledge about different performance modeling and anomaly detection techniques. Section III describes our evaluation framework. Section IV explains the case study setup and proposes three research questions. Sections V, VI, VII investigate the three research questions and present the results. Section VIII discusses the threats to validity. Section IX explains the related work and section X concludes the paper and discusses some future work.

II. Background

Different from functional testing, which has clear pass and fail criteria, there is usually no test oracle available for a load test [8]. The pass/fail criteria of a load test are usually derived based on the "no-worse-than-before" principle. This principle states that the performance of the current version should be at least as good as the previous version(s). In this section, we will briefly explain various test analysis techniques used in a load test. This paper focuses on the automated test analysis techniques using counters. For a survey on automated test techniques using execution logs, please refer to [13].

Each test analysis technique can be further broken down in two parts: (1) *the performance modeling* part (section II-A, in which the system behavior from the past test(s) is summarized into models; and (2) *the anomaly detection* part (section II-B), in which the anomalous behavior in the current test is flagged.

A. Performance Modeling

Table I provides an overview of eight different performance modeling methods. We have divided them into three categories: queuing theory-based methods, data mining-based methods and rule-based methods. For each modeling method, we also list their assumptions and limitations, as well as the data used in the input and output.

Queuing Theory-based Methods: Queuing models apply queuing theory [18] to model the performance behavior of a system. Queuing models mimic system resources (e.g., CPU and memory) as queues, predict the time that different scenarios spend on different queues and estimates the resource utilizations and throughput information. Each scenario corresponds to one type of workload request (e.g., browsing a page or purchasing an item). For each request on each resource, there are two types of timing information that queuing theory tracks and predicts: waiting time and processing time. The waiting time, also called the queuing time, is the time that each request waits in this resource queue to be serviced. The service time, also called the *service demand (SD)*, is the processing time that each request spends on this resource. The overall response time for one request is the combination of the service time and queuing time from all the resources. The resource utilizations and throughput can also be estimated based on the SDs. Since queuing models require the knowledge of the system internals (e.g., the deployment topology, the queues and locks involved in a scenario, and whether the call is synchronous or asynchronous) to properly model the performance behavior, they are also called *white-box models* [19].

In addition to hardware resources, there are also various software resources (e.g., database or middleware) in a system. The single layer queuing models can only be used to model the performance of hardware resources. To model the performance of both hardware and software resources, Layered Queuing Models (LQN) are required [9].

Data Mining-based Methods: Compared to LQN, data mining models treat the system under test as a black box and do not require the knowledge of the system internals. Hence, they are also called *black-box models* [19]. Data mining models rely on various statistical-based and AI-based techniques to model the system's performance behavior. Most of the data mining models used in a load test are regression-based models:

- **Multiple Linear Regression (MLR)**: Linear regression models the system behavior as a linear function of workload mixes and resource usage utilization [10]. As there are certain assumptions associated with linear regression models, extra care is needed to verify these assumptions are met before applying the model. For example, linear regression assumes the input data is normally distributed. If the normality assumption is not met, the input data needs to be transformed (e.g., log transformation) before building the model. In addition, MLR also suffers from confounding problem, which may impact the accuracy of the model. Hence, a set of input variables, which are highly correlated with each other, are removed. This process is called *attribute selection*. In this paper, we evaluate the following two attribute selection approaches: (1) removing the independent variables which are highly correlated with each other [10]; and (2) removing the independent variables which have low correlations with dependent variables [16]. For MLR applied with the first approach, we abbreviate this method as MLR_D; and MLR_V for MLR applied with the second approach.
- **Regression Tree (RegTree)**: Compared to MLR, in which extra care is required to ensure the input data satisfies the assumptions of the model, RegTree does not have such restrictions [16]. RegTree predicts the system's performance behavior using a tree-liked structure.

TABLE I: Performance Modeling Techniques

Category	Techniques	Assumptions	Input	Output	Limitations
Queuing models	LQN Model [9]	System is stable	Workload mix, topology information	Utilization for hardware and software, response time and throughput	Needs topology information
Data mining models	Multiple Linear Regression [10]	Normality, Independence, Linearity, Homoscedasticity	Any counters	Any counters	Assumptions
	MARS [14]	N/A	Any counters	Any counters	N/A
	Quantile Regression [15]	N/A	Any counters	Any counters	Needs to specify quantile
	Regression Tree [16]	N/A	Any counters	Any counters	N/A
Rule-based models	Control Chart [1]	Normality, Linearity, Independence	Any counters	Any counters	N/A
	Descriptive Statistics [11]	N/A	Any counters	Any counters	N/A
	Load Profile [17]	System is stable	Workload mix, topology information	Service demand for each scenario	Needs topology information

- **Multivariate Adaptive Regression Splines (MARS)**: In some cases, a system can undergo different phases during a load test. For example, the performance behavior of a system can be different when the cache is filled up. MARS [14] is proposed to model such behavior (a.k.a., multiple phases), as the simple MLR model cannot perform well in this case. MARS automatically splits the data (a.k.a., phases) and builds different sub-models.
- **Quantile Regression (QUANT)**: Similar to MARS, QUANT [15] can be used to model different phases. Compared to MLR, which imposes equal penalty on overestimation and underestimation, QUANT sets different penalty on overestimation and underestimation based on the given quantiles. These quantiles need to be specified manually in the model. For example, quantile regression built on 25% quantile, denoted as QUANT.25, estimates the relation between input variables and an output variable, on 25% quantile of the output data. In this paper, we have studied the following QUANT models: QUANT.25, QUANT.50, QUANT.75, QUANT.90, QUANT.100. The reason for studying a range of QUANT models is to assess whether different quantiles would impact the effectiveness of detecting performance problems.

Rule-based Methods: Compared to the queuing theory and data mining-based methods, which predict the performance behavior of the current test using the derived models, there are no predictions involved in rule-based methods. Rather they analyze the system behavior using a set of rules.

- **Descriptive Statistics (DescStats)**: Descriptive statistics like median and mean [11] provide a high level summary of the system performance data. They are the easiest to implement and used widely in practice. In this paper, we compare the absolute relative difference between the mean of the current and the previous test(s).
- **Control Chart (CtrlChart)**: A control chart has three components: a control line (CL), a lower control limit (LCL) and an upper control limit (UCL). If a point lies outside the controlled regions (above the UCL or below the LCL), the point is counted as a violation [1]. The three components can be defined in various ways. In this paper, we use the median of the previous test(s) as our CL, the median $\pm$ one standard deviation of the previous test(s) as our UCL and LCL. Similar to MLR, certain assumptions need to be satisfied before applying the CtrlChart.
- **Load Profile (LP)**: The main idea behind LP is that among different versions of the system, the SDs of different requests should be the same. Otherwise, there could be performance problems [17]. There are different approaches to estimate the SDs. In this paper, we use the Kalman filter to estimate the SDs, as it is very flexible (i.e., can work with data from any distributions) and produces high accurate results [20].

B. Anomaly Detection

In general, the data from the current test is considered anomalous, if it "significantly" deviates from the past test(s). The anomaly detection methods are different based on the three categories of performance modeling methods. The anomaly detection methods for the queuing and data mining-based methods are the same, as these methods predict the outcomes based on the input values from the current test. The predicted outcomes are compared against the actual measured values in the current test. The anomaly detection methods for the rule-based methods are different, as the actual values from the current test are evaluated against some rules.

Anomaly Detection Methods for Queuing and Data Mining-based Methods: In this paper, the following two statistical methods are studied. Both methods aim to quantify the differences between the predicted and the measured values:

- **Mean Percentage Absolute Error (MAPE)**: MAPE is one of the evaluation methods of model prediction [21]. The formula to calculate MAPE is shown in Formula 1 below. For each predicted outcome, the absolute relative errors between each predicted and measured values are calculated. Then all these absolute relative errors are combined to provide an average value.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{measured_i - actual_i}{actual_i}\right| \quad (1)$$

- **Wilcoxon Rank Sum test and Cliff's Delta (WC)**: Wilcoxon Rank Sum (WRS) test is a non-parametric test which compares two distributions. However, in some cases, even if the two distributions are different, the differences between them are small. Hence, we also need to quantify the strength of the differences between the two distributions, called the *effect size*. For example, if the response time is more than 5 minutes and the differences of response time between the two tests are very small (e.g., 1 millisecond), the performance deviation from the new test should not be a big concern. Cliff's Delta (CD) is a non-parametric technique to calculate the effect size between two distributions [22]. The strength of the differences and the corresponding range of CD values are shown below:

$$\text{effect size} = \begin{cases} \text{trivial} & \text{if } CD < 0.147 \\ \text{small} & \text{if } 0.147 \leq CD < 0.33 \\ \text{medium} & \text{if } 0.33 \leq CD < 0.474 \\ \text{large} & \text{if } 0.474 \leq CD \end{cases}$$

Anomaly Detection Methods for Rule-based Methods: The anomaly detection methods for rule-based methods are mainly based on threshold values. For example, if more than 30% of the CPU utilizations data from the current test are violations (above UCL or below LCL), the CPU metric in the current test is considered anomalous [1].

III. An Overview of Our Framework

In this section, we will introduce our framework to evaluate the effectiveness of different test analysis techniques as illustrated in Figure 1. Our framework consists of the following five steps: (1) load testing; (2) model building; (3) test oracle derivation; (4) anomaly detection & result evaluation, and (5) ranking. Our framework is very flexible, as it can easily accommodate new test analysis techniques, new load testing data, and new evaluation systems.

Step 1 - Load Testing

Multiple versions of the same system are load tested. The selected versions to be tested should be a mixture of "good" and "bad" versions. Bad versions contain known performance problems. Good versions refer to the versions of the system whose performance is satisfactory under load. All the selected versions of the same system are load tested using the same load profile. A load profile is characterized along two dimensions: workload mixes and workload intensities. The *workload mix* refers to the ratios among different types of requests (e.g., 30 percent browsing, 10 percent purchasing and 60 percent searching). The *workload intensity* refers to the rate of the incoming requests (e.g., browsing, purchasing and searching), or the number of concurrent users. During the course of each load test, while the system is processing requests generated by the load profile, the system behavior data (a.k.a., logs and counters) is collected. In this paper, we use the load profile, which generates stable request rate, to test the target systems.

This same load profile is applied on multiple versions of the system to counter potential bias. The test plan will be updated if the system evolves. For example, if the URL of the browsing scenario changed from "brs.jsp" to "browse.jsp" in the next version, the test plan will be updated accordingly.

Step 2 - Model Building

Using the generated system behavior data from the previous step, different performance modeling methods are applied to summarize the performance behavior of the previous test(s). For descriptions of different performance modeling methods, please refer to Section II-A. Additional data processing is carried out to ensure the data is cleaned up (e.g., removing the warm-up and cool-down period) and transformed into the corresponding input file formats (e.g., csv or xml) required by different modeling techniques.

In addition, extra care is taken to ensure the assumptions are met for each model. For example, MLR and CtrlChart require input data to be normally distributed. Hence, we first use Shapiro-Wilk test to verify the normality assumption. If the normality assumption is not met, log transformation is performed on the input data. For QUANT, we choose a range of different quantizes: 25%, 50%, 75%, 90% and 100%, to investigate if models built on different quantiles performed the same . In this paper, we use the readily available OPERA tool [23] as our model implementation for LQN and LP methods. OPERA automatically estimates the SD using Kalman filter based on the workload mixes, hardware resource utilizations and response time for different scenarios. OPERA models and predicts the system performance using the Mean-Value-Analysis (MVA) algorithm. All the other methods are re-implemented using R.

Step 3 - Test Oracle Derivation

In order to evaluate the effectiveness of different test analysis techniques, a "test oracle", which flags the problematic counters in each run, is needed. We manually derive the test oracle for each run based on bug reports, release notes and close examinations on the counters. For example, if a version of a system injected with a CPU intensive bug, the CPU utilization of this version should be flagged as "buggy". As another example, if the response time for the browsing scenario is much longer in the current version than the previous version(s) and there is a response time issue reported in the bug report for this version, the response time for the browsing scenario is also flagged as "buggy". Our test oracle has been derived and verified by the first two authors of this paper.

Step 4 - Anomaly Detection & Model Evaluation

Using the derived performance models in the previous step, different anomaly detection techniques are applied to flag suspicious behavior from the current test. Please refer to Section II-B for descriptions on different anomaly detection methods. All the anomaly detection methods are re-implemented in R. Different test analysis techniques may use different threshold values to detect anomalous counters:

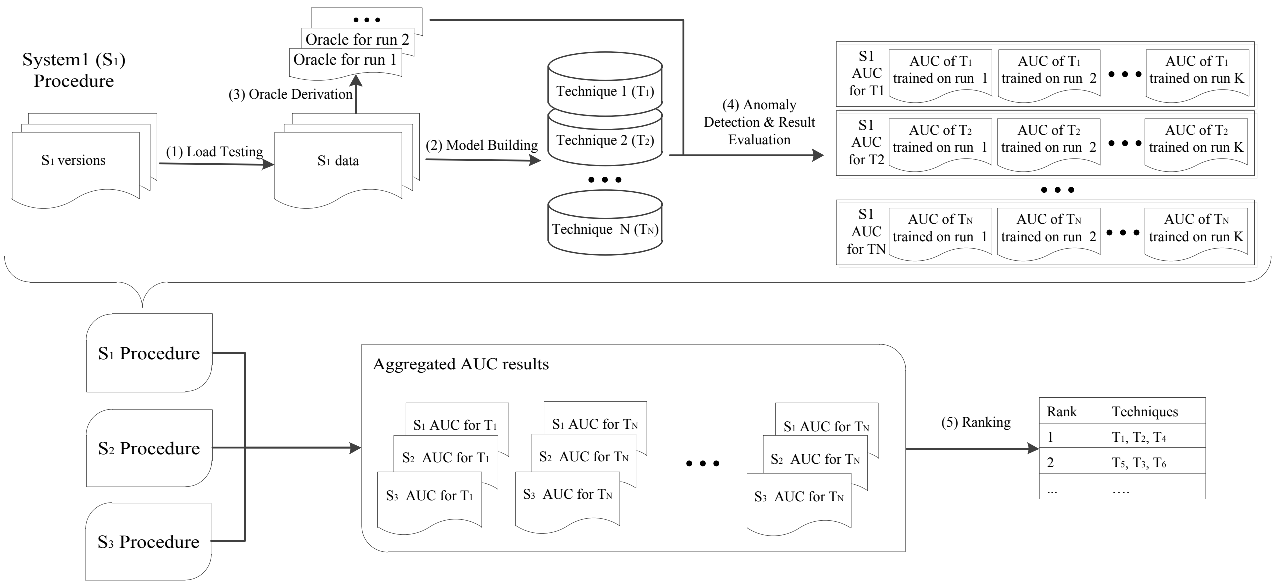

Fig. 1: An Overview of Our Framework

- **MAPE**: The higher the MAPE values are, the bigger the deviations between the predicted and the measured values in the current test. Ideally, higher MAPE values for one counter indicate higher probability that this counter is anomalous. However, if the MAPE cut-off value is set to be too high, we might miss some problematic counters.
- **WC**: For the WC method, the threshold values are only used when there is a statistical difference between the current and the past test(s). The strength of the differences are quantified by the CD values. The counters with higher CD values, indicating bigger differences, and are more likely to be problematic.
- **CtrlChart**: For CtrlChart, the threshold value corresponds to the violation ratio, which is the percentage of data points in the current test lying outside the control limit (a.k.a., above the UCL or below the LCL). Higher violation ratios indicate bigger differences between the current and past test(s).
- **DescStats**: For DescStats, the threshold value corresponds to the absolute percentage errors between the current test and the previous test(s).

A test analysis technique corresponds to one combination of a modeling technique and an anomaly detection techniques. The predicted values from each performance modeling method are evaluated using all the applicable anomaly detection methods to ensure all valid test analysis techniques are studied. For example, the predicted values from the RegTree are applied using both MAPE and WC. There are ten different data mining and queuing methods (LQN, MLR_D, MLR_R, MARS, RegTree, and 5 different QUANT methods), each of which has two different anomaly detection methods (WC and MAPE). In addition, we have three different rule-based performance modeling methods, each of which has one anomaly detection method. Hence, we have evaluated 23 different testing analysis techniques in total.

We use the Receiver Operating Characteristic (ROC) curve [24] to assess the effectiveness of different test analysis techniques under a range of threshold values. Once the ROC curve for each test analysis technique is built, the AUC (Area Under the Curve) values are calculated. The AUC values are used in the ranking process below.

Step 5 - Ranking

In this paper, we use the Scott-Knot test [25] to rank the effectiveness of different test analysis techniques. Scott-Knott test uses hierarchical clustering to group the test analysis techniques. Each group contains techniques with similar performance. The smaller the group number that one technique is assigned to, the more effective this test analysis technique is. For example, if CtrlChart is assigned to Group 1, it means that CtrlChart is more effective than techniques assigned in Groups 2 and 3. Scott-Knot test has been used successfully in other software engineering tasks (e.g., comparing the performance of different bug prediction techniques [26]). The input for the Scott-Knott test is the AUC values from different test analysis techniques exercised on different tests.

There are two reasons for using Scott-Knot test and the AUC values to rank the test analysis techniques. First of all, compared to other techniques (e.g., Nemenyi's test or hierarchical clustering), Scott-Knot test automatically clusters different test analysis techniques into distinctive groups, which are non-overlapping. Second, most test analysis techniques

TABLE II: Case Study Systems

System	Domain	Category	SLOC
DS2	Benchmark application	JSP	100
Petclinic	E-commerce	Hibernate	$>$ 2000
JMS	Mail Server	Mailet Container	$>$ 4,000

TABLE III: Load Testing Details for Three Systems

System	# of Test Scenarios	Versions	# of Test Runs	Injected/Known Bugs or Improvements
DS2	4	v0	6	Original version
		v1	1	Injected a CPU intensive bug in purchasing
		v2	1	Removed indices in two database tables
		v3	1	Created a constant delay in purchasing
PetClinic	15	v0	1	Original version
		v1	1	Changed to stateless and non-blocking I/O to improve request throughput
		v2	6	Removed JVM locks to further improve request throughput
JMS	6	2.3.2	1	Stable release version
		3.0M1	1	Improved the efficiency of queueing emails
		3.0M4	6	Improved the efficiency of handling MIME messages

require threshold values, which are either not specified in the paper, or are tunable based on different projects. AUC values enable us to evaluate the effectiveness of different techniques under a range of threshold values.

IV. Case Study Setup

We intentionally picked three different open source systems with different deployment setup and application domains: Dell DVD Store (DS2) [27], Petclinic [28] and James Mail Server (JMS) [29] to demonstrate the usefulness of our framework. Each system is developed under different technologies and has different deployment configurations. Table II shows the details.

In total, about 60 hours of load testing were run on various versions of the three systems. We collected over 150 counters from three target systems, containing resource utilizations (e.g., CPU, memory and disk I/O) for various components of the systems, request rates, throughput and response time for each scenario.

The rest of this section is organized as follows: section IV-A explains the test scenarios as well as the rationales for selecting different versions. Section IV-B describes our test execution and test monitoring process. Section IV-C introduces a few research questions that we want to answer using our framework.

A. Test Design

1) DS2: DS2 is an open source e-commerce website, which Dell uses to benchmark the performance of their hardware. The system contains four different JSP pages: login, browsing catalogs, purchasing items and registrations for new customers. It is deployed on the top of a Tomcat web server and is connected to a MySQL database.

The test details are shown in Table III. The original version (v0) does not have any injected or known performance bugs, so we call it “good version”. V0 was load tested six times. We injected three performance bugs (a busy CPU bug, a database performance bug, and a constant delay bug) into three versions of DS2 (v1, v2, and v3), respectively. The CPU and the database bugs are from [1]. We have added the delay bug in this study, as it represents another performance issue (a.k.a., blocking v.s. busy waiting). These three versions are called “bad versions”. Following the practice of [10], in which a good version is load tested multiple times and each bad version is tested once. If a load test is executed on a good version, we call it a “good run”, and a “bad run” means a load test is executed on a bad version.

PetClinic: Petclinic is a web application written in Java Spring and Hibernate. There are 15 different test scenarios related to pets, pet owners, and veterinarians. Example scenarios include scheduling visits to veterinarians and locating pet owners. Same as DS2, Petclinic deploys on top of a Tomcat web server and connects to a MySQL database.

Different from DS2, in which performance bugs are manually injected, we load tested on versions of systems with known real-world performance issues for PetClinic. In the blog post by Dubois [30], he detailed a series of changes that he has made in order to improve the performance and scalability of PetClinic. All his changes are archived in the GitHub repository [31]. In this paper, we picked three versions from that GitHub repository. We called the last version as the “good version”, as it contains all the fixes that he has performed. The other two older versions as “bad versions”, as they contain various performance issues. The descriptions for the performance issues associated with the bad versions are shown in Table III. We executed the load testing on good version 6 times and on each bad version once.

JMS: The Apache James Mail Server (JMS) is an open source, mail server written in Java. It contains two generic scenarios: sending emails and receiving emails. The two generic scenarios can be further divided into many smaller scenarios, like sending and receiving emails with or without attachments, reading only the mail header or loading the whole mail, etc. In total, 6 scenarios are tested for JMS. Different from DS2 and PetClinic, JMS is a stand-alone system, which does not have any additional supporting components.

Unlike Petclinic, we could not find two or more continuous revisions, which contain performance fixes. Hence, we picked three different release versions in our study. Apart from the latest stable version 2.3.2, there were three MR releases for version 3.0. Since 3.0M3 was not available to download in the official website, the other two MRs, 3.0M1 and 3.0M4, are used in this study. The selected three versions have many bug fixes and performance improvements. Between these versions, numerous modules have been upgraded, such as updating to more efficient JavaMail [32] and ActiveMQ versions [33]. Table III summarizes the various performance bug fixes and improvements from the two released versions. This information is extracted from JMS’s bug reports and release notes. Similar to DS2 and PetClinic, since 3.0M4 (the "good version") is the last version containing most performance fixes, it was load tested 6 times. The other two versions (the "bad versions") were tested once.

B. Test Execution and Monitoring

All three systems are deployed on the same machine, which has the following hardware configurations: Intel® Core™ 2 CPU 2.40 GHz, 2 GB memory and a 160 GB 7200 RPM hard drive. We use JMeter [34] as our load generator. JMeter is deployed on a separate machine with the following hardware specifications: Intel® Core™ i7-4790 CPU 3.60 GHz, 16 GB memory and a 2TB 7200 RPM hard drive. The reason for the separate deployment of JMeter and the case study systems is to ensure no overhead caused by the load generator to the case study systems.

The resource utilizations for the systems during load testing are monitored and recorded using pidstat [35]. In addition, we also archived the logs from JMeter, MySQL and Tomcat at the end of each load test. These logs can be used to extract the workload and response time information.

C. Research Questions

We propose the following three research questions to evaluate the effectiveness of different test analysis techniques:

- **RQ1 (Build Dependency)**: *Can we use data from a bad run to analyze the results of other tests?*
  In some cases, there could be no "good versions" available for a system, as the system is constantly being fixed and improved until release. Hence, most of the load testing runs could be "bad runs". In this research question, we seek to answer whether we can still leverage the data from the bad runs to effectively analyze the results of a load test. RQ1 is discussed in Section V.
- **RQ2 (History Dependency)**: *Would more historical data help improve the performance of different test analysis techniques?*
  Since all the test analysis techniques compare the current test data against the historical data (a.k.a., past runs). This research question aims to answer whether more historical data would help improve the effectiveness of different test analysis techniques. RQ2 is discussed in Section VI.
- **RQ3 (Sampling Interval Dependency)**: *What's the impact of different sampling intervals on detecting performance problems?*
  During the course of a load test, the behavior of the system is monitored. There can be different monitoring granularities. For example, the average CPU utilizations can be sampled at every 5 seconds interval or every 60 seconds interval. Different sampling intervals generate different levels of details and different sizes of data to be analyzed. This research question aims to answer whether more fine-grained sampling intervals would improve the effectiveness of different test analysis techniques. RQ3 is discussed in Section VII.

TABLE IV: Aggregated Scott-Knot Ranking for RQ1

Category	**Techniques**	**Rankings**			
		G		B	
Rule-based Models	CtlChart	1		1	
	DescStats	1		1	
	LP	2		2	
		M_B	M_G	W_B	W_G
Data Mining Models	MLR_D	2	2	2	2
	MLR_V	2	2	2	2
	MARS	2	2	2	2
	QUANT.25	2	2	2	2
	QUANT.50	2	2	2	2
	QUANT.75	2	2	2	2
	QUANT.90	2	2	2	2
	QUANT.100	2	2	2	2
	RegTree	1	1	1	1
Queuing Models	LQN	2	2	2	2

V. RQ1 - Build Dependency

In practice, it can be hard to obtain one version of a system which is "bug free" due to the following two main reasons: (1) constant evolution of the system due to bug fixes and feature improvements; and (2) undetected performance bugs due to the challenging nature of analyzing the results of a load test [13]. Hence, in this RQ, we want to investigate whether we can still leverage the data from "bad runs" to effectively analyze the results of current load test.

Approach

We built a set of performance models, which use the data from one good run or one bad run. These performance models are used to evaluate the test results from other test runs. For example, eight LQN techniques are built using the test runs from PetClinic. Among them, four techniques are built using the data from bad versions (v0 and v1 of PetClinic) and the rest four techniques are built using the data from good versions (v2 of PetClinic). Each model is used to predict the results of all the other test runs from PetClinic. Similar process is applied on other techniques and on other systems. The prediction results from different test runs of the same test analysis techniques are grouped. The overall results across all systems are ranked using the Scott-Knot test.

Analysis

Table IV shows the ranking results for different test analysis techniques. The second column shows the list of performance modeling methods. Each cell corresponds to the ranking of one particular test analysis technique, as technique corresponds to one unique combination of a modeling method and an anomaly detection method. The columns marked with "B" and "G" correspond to the techniques built with training data from the good and bad runs, respectively. All the queuing and data mining methods are evaluated under two anomaly detection methods: MAPE (marked as "M" in the table) and WC (marked as "W"). Hence, the column "M_B" refers to the modeling techniques trained with data from the bad run and use MAPE as its anomaly detection method.

There are two groups generated by the Scott-Knott test. Among all the test analysis techniques, DescStats, CtrlChart, both RegTree techniques perform better than the other techniques. Among all the test analysis techniques, their rankings are the same regardless of using data from the good or the bad runs. We want to note that some performance models do not summarize the training well. For example, when we apply the LQN model built from the bad runs (e.g., v3 from DS2) to predict the values from the same run (v3), the accuracy is low. However, when the LQN model built from the good runs (e.g., v0 from DS2) to predict the values from the same run (v0), the accuracy is high. However, in general, the accuracy of the modeling methods does not seem to impact the effectiveness of different test analysis techniques. We believe this is because each technique can effectively detect the performance deviations between the good and the bad runs.

Findings: In general, the effectiveness of different test analysis techniques are not impacted by its training data.
Implications: Load testing practitioners can use the test analysis techniques as an effective "diffing" tool to check if there are any performance deviations between the current and the past test(s).

VI. RQ2 - History Dependency

All the studied load testing analysis techniques compare the data from the current test against the historical data (a.k.a., past runs) to ensure the performance of the current test is not worse-than the past test(s). As there can be many past test(s), the time required for load testing analysis can be increased due to the number of past test(s) that need to be compared. If adding more past tests could provide a significant boost for the effectiveness of detecting performance bugs, we need to know the number of past test(s) required. However, if we can achieve the same effectiveness by analyzing fewer past test(s), we can save the limited test analysis time. The goal of this RQ is to investigate whether more historical data could help improve the effectiveness of different test analysis techniques.

Approach

The previous RQ shows that all the test analysis techniques are effective regardless of building performance models using good or bad runs. In this RQ, we built a set of performance models with varying number of good runs as their training data for each test analysis technique. For example, we combined two different good runs as training data. This process was repeated six times to ensure each good run can appear at least once in one of the six training datasets. The same approach is repeated for combining 3–5 good runs. In the end, there are 30 different training sets for each modeling technique. These models are used to detect performance problems for the other test runs. For example, if the model is built using data from three good runs in JMS, the anomaly detection methods would be applied on the other 3 good runs and 2 bad runs. The same process is applied on all three systems.

Analysis

Table V shows the ranking results for different test analysis techniques based on varying number of previous tests as their training data. Similar to the previous RQ, we compared 23 different test analysis techniques based on their performance modeling and anomaly detection techniques. The second column corresponds to different performance modeling techniques. The capital letter "R" refers to the number of good runs used in the training data. For example, "1R" means 1 good run is used to build the model, and "2R" means 2 good runs are used to build the model, and so on. Each cell stores the ranking for each test analysis technique. "M" and "W" refer to the MAPE and WC anomaly techniques, respectively. "M_1R" refers to using 1 good run as the training data and MAPE as its anomaly detection method.

As we can see in Table V, there are total five groups. Among all the test analysis techniques, CtrlChart, DescStats and (RegTree + WC) are ranked the top. All the test analysis techniques share similar trends based on the amount of training data: (1) each technique performs at least the same or better giving more previous test(s) as their training data; and (2) the effectiveness of the test analysis techniques seems to be stabilizing after 2R.

Findings: The test analysis techniques perform better by providing more previous tests as their training data. Based on our load testing data, the effectiveness of the test analysis techniques seems to stabilize after using two previous test run(s) as their training data.
Implications: Comparing the current test against multiple previous test runs is time-consuming and might not improve the effectiveness of detecting performance bugs. More research is required to investigate the optimal amount of data required for training the performance models.

VII. RQ3 - Sampling Interval Dependency

During the course of the load test, the system behavior is monitored and recorded periodically. For example, the average CPU and memory utilization can be sampled at every 5 seconds, 30 seconds, 60 seconds or more. The sampling interval affects the resulting test data in terms of size and granularity. As the test duration is fixed for one load test, the amount of recorded data decreases when the sampling interval is large. However, it might lose some details. For example, a sudden spike in the CPU could be visible using the 5 seconds sampling interval but might not be visible using the 180 seconds sampling interval. However, using a smaller sampling interval generates larger volumes of data to be analyzed and could slow down the system execution. For example, it is usually not recommended to set the sampling interval for pidstat to be less than 5 seconds [36], as the monitoring overhead is too high and significantly slows down the test execution. Such slow-down could impact the validity of the collected testing data, as the system's performance no longer reflects the actual behavior in the field. In this section, we want to examine the impact of different sampling intervals on the effectiveness of test analysis techniques.

Approach

The load tests conducted for the three systems were monitored using 5 seconds as its sampling interval. For all the

TABLE V: Aggregated Scott-Knot Ranking Results for RQ2

Category	Techniques	Rankings									
		1R		2R		3R		4R		5R	
	CtlChart	1		1		1		1		1	
Rule-based Models	DescStats	1		1		1		1		1	
	LP	5		5		5		5		5	
		M_1R	M_2R	M_3R	M_4R	M_5R	W_1R	W_2R	W_3R	W_4R	W_5R
	MLR_D	3	3	3	3	3	3	3	2	2	2
	MLR_V	3	3	3	3	3	3	3	3	3	3
	MARS	3	3	3	3	3	3	3	2	2	2
	QUANT.25	3	3	3	3	3	4	3	3	3	3
Data Mining Models	QUANT.50	4	3	3	3	3	4	3	3	3	3
	QUANT.75	4	3	3	3	3	4	3	3	3	3
	QUANT.90	4	3	3	3	3	4	3	3	3	3
	QUANT.100	3	3	3	3	3	3	3	3	3	3
	RegTree	2	2	2	2	2	1	1	1	1	1
Queuing Models	LQN	3	3	3	3	3	3	3	3	3	3

systems, we aggregate the testing data into four additional granularities: 30, 60, 180 and 300 seconds to represent the data collected at 30, 60, 180 and 300 seconds sampling intervals. For each sampling interval, multiple performance models, which use the data from one good run, are built. The resulting models are used to analyze the rest of test runs.

Analysis

Table VI shows the ranking results for different test analysis techniques based on the test runs monitored under different sampling intervals. Similar to the previous two RQs, the second column shows different performance modeling techniques. The numbers after the letter "i" indicate the sampling intervals. For example, "i5" means data collected using 5 seconds as its sampling interval, "i30" means data collected using 30 seconds as its sampling interval, and so on. "M_i5" refers to using the data collected at 5 seconds sampling interval as the training data and MAPE as its anomaly detection method.

As we can see in Table VI, there are four groups outputted by the Scott-Knott tests. Similar to the previous two RQs, CtrlChart, DescStat and two RegTree techniques rank the best. For each technique, surprisingly, using data from smaller sampling intervals usually performs the same or worse than using data from larger sampling intervals. In general, the test analysis techniques perform the best when using data sampled at every 180 seconds or more. We suspect this is because the data trend is smoother under larger sampling intervals.

Findings: For most of the analysis techniques, using the data collected at smaller sampling intervals yields worse performance than larger sampling intervals. The techniques perform the best when using data sampled at $\geq$ 180 seconds.
Implications: Fine-grained monitoring could bring overhead to the system execution and does not necessarily improve the effectiveness of different test analysis techniques. Larger sampling interval may miss some of the runtime behavior. More research is required to investigate the optimal sampling interval for load testing analysis.

VIII. Threats To Validity

In this section, we will discuss the threats to validity.

A. Construct Validity

Sampling interval: In this paper, we evaluated different test analysis techniques under different sampling intervals. All the test runs are monitored using pidstat under the 5 seconds sampling interval. We aggregated the pidstat values in RQ3 to simulate the pidstat data generated under the 30, 60, 180, and 300 seconds interval. We also ran one load test using the 30 seconds sampling interval and compared the resulting data against our aggregated data. The differences are negligible.

The accuracy of the performance data: Each load test was run for a few hours to avoid measurement bias and errors [37].

The resource utilizations are collected using pidstat. The throughput and the response time data from each scenario are extracted either from the JMeter or Tomcat access logs. We cross-checked the data extracted from JMeter and Tomcat and found they would agree most of the time. Hence, we used the data extracted from access logs for DS2 and PetClinic, and used JMeter logs for JMS as JMS doesn't use Tomcat.

B. Internal Validity

For MLR and CtrlChart, we performed log transformation to ensure the model assumptions are met before building the model. In addition, to avoid the confounding problem, we also applied two variable section techniques [10], [16] for MLR to reduce the number of input variables. We used the Kalman filter to estimate the SD for different scenarios. The values of SD are used in the LQN and LP models. Compared to other SD estimation techniques, studies show that Kalman filter is the least restrictive (a.k.a., works on data from any distributions) and produces high accurate results [20].

For each RQ, we keep all two of the three factors (good or bad runs, number of past tests, sampling intervals) the same while investigating the impact of one factor in order to avoid the risk of confounding. For example, in RQ1, we keep the number of past tests and the sampling intervals the same, while varying the types of training data (good run or bad run).

C. External Validity

Most of the previous load testing research is usually conducted on industry systems. Hence, the testing data from such systems are rarely available to download due to confidentiality

TABLE VI: Aggregated Scott-Knot Ranking Results for RQ3

Category	Technique	Rankings									
		i5		i30		i60		i180		i300	
Rule-based Models	CtlChart	2		1		1		1		1	
	DescStats	1		1		1		1		1	
	LP	4		4		4		4		4	
		M_i5	M_i30	M_i60	M_i180	M_i300	W_i5	W_i30	W_i60	W_i180	W_i300
Data Mining Models	MLR_D	3	2	2	2	2	2	2	2	2	2
	MLR_V	3	2	2	2	2	2	2	2	2	2
	MARS	3	2	2	2	2	2	2	2	2	2
	QUANT.25	3	2	3	2	2	3	3	3	2	2
	QUANT.50	3	2	3	2	2	2	3	3	2	2
	QUANT.75	3	3	3	2	2	3	3	3	2	2
	QUANT.90	3	3	3	2	2	3	3	3	2	2
	QUANT.100	3	3	2	2	2	3	3	2	2	2
	RegTree	2	2	1	1	1	3	1	1	1	1
Queuing Model	LQN	3	2	2	2	2	3	2	2	2	2

concerns. In this paper, we have evaluated different test analysis techniques on multiple versions of three open source systems. These three systems are from different domains (e.g., benchmarking, e-commerce and mail servers) and are implemented using different technologies. In addition, when selecting the versions of the systems to be load tested on, we have used two approaches: (1) for DS2, we manually injected a few performance bugs, which are also used in previous studies (e.g., [1], [10]); (2) for PetClinic and JMS, we have selected the versions which contain real-world performance bugs. However, our findings in three RQs may not generalize to other systems. For example, we have applied the stable load profiles for one system. Other systems may be load tested using bursty load profiles. In this case, some test analysis techniques may not perform well. However, our proposed evaluation framework is very flexible, new test data (from the same or different load profiles or systems) can be easily incorporated and compared in the framework.

IX. Related Work

Here we discuss the prior research related to this paper.

A. Load Testing

In general, a load test can be broken down into three phases: test design, test execution and test analysis. For a survey on various techniques used in a load test, please refer to [13].

In general, there are two types of load profiles that can be used in a load test: *realistic testing loads* and *fault-inducing testing loads*. Realistic testing loads aim to generate load profiles which mimic the actual request rates in the field [11]. This type of load profiles are usually derived from past usage data (e.g., Tomcat access logs). Compared to realistic testing loads, which require repeatedly executing the same scenarios for a long duration of time (hours or days), the fault-inducing load profiles aim to generate load profiles, which likely lead to performance problems. For example, Grechanik et al. [38] propose an iterative technique which automatically learns the system behavior using randomly selected inputs. Then machine learning techniques are applied to generate inputs which lead to performance problems. Zhang et al. [39] use Java PathFinder to automatically generate load profiles, which cause performance problems. They assign a time value for each step along the code path. By summing up the costs for each code path, they can identify the paths that lead to the longest response time. The values that satisfy the path constraints form the load profiles. In this paper, we focus on evaluating different test analysis techniques, which analyze the data generated from realistic testing loads.

B. Evaluation

Although this is the first work which systematically evaluates different load testing analysis techniques, there have been a few prior works which compare the techniques used to solve other software engineering problems. For example, Lutellier et al. [40] compare the accuracy of different architecture recovery techniques. Ghotra et al. [26], [41] compare the performance of different bug prediction techniques. Bellon et al. [42], [43] compare the accuracy of different clone detection techniques. Parnin and Orso [44] empirically evaluate different automated debugging techniques. Lo et al. [45] compare the effectiveness of different fault-localization techniques.

X. Conclusion and Future Work

In this paper, we have proposed an evaluation framework which systematically compares the effectiveness of different load testing analysis techniques. To demonstrate the usefulness of our framework, we have applied it on the load testing data exercised on 25 runs of three open source systems (DS2, PetClinic and JMS). We have found that all the techniques perform well regardless of training on good or bad test runs. Using more than two runs of previous tests as the training data takes longer time to analyze and may or may not yield better performance. The test analysis techniques are more effective when building and analyzing data from higher sampling intervals ($\geq$ 180 seconds). Among all the techniques, CtrlChart, DescStats, and RegTree are the most effective ones.

In the future, we plan to extend our evaluation study using load testing data from additional systems as well as incorporating new test analysis techniques. In addition, we also plan to evaluate the test analysis techniques along other dimensions (e.g., usefulness to practitioners).

References

[1] 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 *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, 2012.

[2] M. D. Barros, J. Shiau, C. Shang, K. Gidewall, H. Shi, and J. Forsmann, "Web Services Wind Tunnel: On Performance Testing Large-Scale Stateful Web Services," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007.

[3] E. J. Weyuker and F. I. Vokolos, "Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study," *IEEE Transactions on Software Engineering*, 2000.

[4] "Steve Jobs on MobileMe," http://arstechnica.com/journals/apple.ars/2008/08/05/steve-jobs-on-mobileme-the-full-e-mail, visited 2015-10-23.

[5] S. G. Stolberg and M. D. Shear, "Inside the Race to Rescue a Health Care Site, and Obama," 2013, http://www.nytimes.com/2013/12/01/us/politics/inside-the-race-to-rescue-a-health-site-and-obama.html, visited 2015-10-23.

[6] A. Avritzer and E. J. Weyuker, "The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software," *IEEE Transactions on Software Engineering*, 1995.

[7] C.-W. Ho, L. Williams, and B. Robinson, "Examining the Relationships between Performance Requirements and "Not a Problem" Defect Reports," in *Proceedings of the 2008 16th IEEE International Requirements Engineering Conference (RE)*, 2008.

[8] A. Avritzer and A. B. Bondi, "Resilience Assessment Based on Performance Testing," in *Resilience Assessment and Evaluation of Computing Systems*, K. Wolter, A. Avritzer, M. Vieira, and A. van Moorsel, Eds. Springer Berlin Heidelberg, 2012.

[9] C. Barna, M. Litoiu, and H. Ghanbari, "Autonomic load-testing framework," in *Proceedings of the 8th ACM international conference on Autonomic computing*, 2011.

[10] W. Shang, A. E. Hassan, M. Nasser, and P. Flora, "Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015.

[11] D. A. Menasce, "Load Testing, Benchmarking, And Application Performance Management For The Web," in *Proceedings of the 2002 Computer Management Group Conference (CMG)*, 2002.

[12] R. Gao, Z. M. J. Jiang, C. Barna, and M. Litoiu, "Replication Package," https://www.dropbox.com/s/1xzfxc28pf2h284/rep_package.zip?dl=0.

[13] Z. M. Jiang and A. E. Hassan, "A Survey on Load Testing of Large-Scale Software Systems," *IEEE Transactions on Software Engineering*, 2015.

[14] M. Courtois and M. Woodside, "Using Regression Splines for Software Performance Analysis," in *Proceedings of the 2nd International Workshop on Software and Performance (WOSP)*, 2000.

[15] A. B. de Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Why you should care about quantile regression," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[16] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vPerfGuard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013.

[17] 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 *2013 17th European Conference on Software Maintenance and Reengineering (CSMR)*, 2013.

[18] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[19] D. Didona, F. Quaglia, P. Romano, and E. Torre, "Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2015.

[20] T. Zheng, M. Woodside, and M. Litoiu, "Performance Model Estimation and Tracking Using Optimal Filters," *IEEE Transactions on Software Engineering*, 2008.

[21] A. Heiat, "Comparison of artificial neural network and regression models for estimating software development effort," *Information and software Technology*, 2002.

[22] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?" in *Annual meeting of the Florida Association of Institutional Research*, 2006.

[23] M. Litoiu and C. Barna, "A performance evaluation framework for Web applications," *Journal of Software: Evolution and Process*, 2013.

[24] S. Wu and P. Flach, "A scored AUC metric for classifier evaluation and selection," in *Second Workshop on ROC Analysis in ML, Bonn, Germany*, 2005.

[25] E. G. Jelihovschi, J. C. Faria, and I. B. Allaman, "ScottKnott: a package for performing the Scott-Knott clustering algorithm in R," *TEMA (São Carlos)*, 2014.

[26] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models," in *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, 2015.

[27] "Dell DVD Store Database Test Suite," http://linux.dell.com/dvdstore/, visited 2015-10-23.

[28] "A sample Spring-based application," https://github.com/spring-projects/spring-petclinic, visited 2015-10-23.

[29] "James Project," http://james.apache.org/, visited 2015-10-23.

[30] J. Dubois, "Spring PetClinic Sample Application," http://blog.ippon.fr/2013/03/11/improving-the-performance-of-the-spring-petclinic-sample-application-part-1-of-5/, visited 2015-10-23.

[31] ——, "Spring PetClinic Sample Application," https://github.com/jdubois/spring-petclinic, visited 2015-10-23.

[32] "JavaMail," http://www.oracle.com/technetwork/java/javamail/index.html, visited 2015-10-23.

[33] "Apache ActiveMQ," http://activemq.apache.org/, visited 2015-10-23.

[34] "Apache JMeter," http://jmeter.apache.org/, visited 2015-10-23.

[35] "Performance monitoring tools for Linux," https://github.com/sysstat/sysstat, visited 2015-10-23.

[36] "Advanced Load Testing Features of Visual Studio Team System," http://blogs.msdn.com/b/billbar/archive/2006/01/24/advanced-load-testing-features-of-visual-studio-team-system.aspx, visited 2015-10-23.

[37] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically Rigorous Java Performance Evaluation," in *Proceedings of the 22nd International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, 2007.

[38] M. Grechanik, C. Fu, and Q. Xie, "Automatically Finding Performance Problems with Feedback-directed Learning Software Testing," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, 2012.

[39] P. Zhang, S. Elbaum, and M. B. Dwyer, "Compositional Load Test Generation for Software Pipelines," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA)*, 2012.

[40] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidović, and R. Kroeger, "Comparing Software Architecture Recovery Techniques Using Accurate Dependencies," in *Proceedings of the 37th International Conference on Software Engineering (ICSE) - Volume 2*, 2015.

[41] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, 2012.

[42] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and Evaluation of Clone Detection Tools," *Software Engineering, IEEE Transactions on*, 2007.

[43] J. Svajlenko, C. K. Roy, and J. R. Cordy, "A Mutation Analysis Based Benchmarking Framework for Clone Detectors," in *Proceedings of the 7th International Workshop on Software Clones (IWSC)*, 2013.

[44] C. Parnin and A. Orso, "Are Automated Debugging Techniques Actually Helping Programmers?" in *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA)*, 2011.

[45] T.-D. B. Le, D. Lo, and F. Thung, "Should I follow this fault localization tool's output?" *Empirical Software Engineering*, 2015.