1 SYSTEMATIC MAPPING STUDY

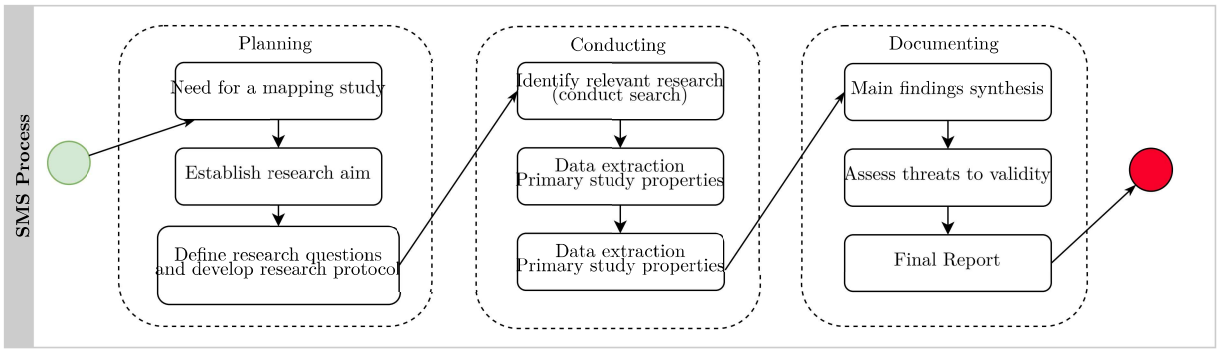
This chapter describes the procedures used to investigate the literature for this study. For this purpose to be achieved was carried out the planning and execution of a Systematic Mapping Study (SMS).

1.1 Protocol

An SMS identifies, selects and critically appraises research in order to answer a clearly formulated question (BARN; BARAT; CLARK, 2017). The SMS should follow a clearly defined protocol or plan where the criteria is clearly stated before the mapping is conducted. It is a comprehensive, transparent search conducted over multiple databases that can be replicated and reproduced by other researchers. It involves planning a well thought out search strategy which has a specific focus or answers a defined question. The review identifies the type of information searched, critiqued and reported within known time-frames.

The SMS is built on a structured and defined process presented by Engström e Petersen (2015). This process defines the necessary steps to achieve the SMS objectives. Figure 1 shows the steps of the SMS as well as the tasks that were performed in the work conduction.

Figure 1 – SMS Process



Source: Adapted from Engström e Petersen (2015).

1.1.1 Scope and Objective

With the purpose of provide an empirical reference for professionals and researchers who search for new tools or tools that have certain particularities in the development and execution of performance testing, the objective of this study is to identify and characterize existing performance testing tools in the literature. In addition, we aim at identifying the academic and open source tools and finding out their quality attributes. In this sense, the description of the goal is described according to the GQM (Goal, Question, Metric) paradigm Koziolek (2008) can be observed in Table 1.

Table 1 – Objective according to the GQM paradigm

For the purpose of:	*Identify / Characterize*
With respect to:	*performance testing tools*
From the viewpoint of:	*performance test engineers and researchers*
In the context of:	*performance testing environment*

1.1.2 Question Structure

The Research Questions (RQs) are structured based on the Population, Intervention, Comparison, Outcome and Context (PICOC) criteria as recommended by Kitchenham (2007) and can be observer in Table 2.

Table 2 – PICOC structure

Population:	*published research on software;*
Intervention:	*performance testing;*
Comparison:	*general comparison of the retrieved tools;*
Outcome:	*performance testing tools;*
Context:	*both academic and industrial context.*

1.1.3 Research Question

The following RQs are define according to the PICOC question structure established in Section 1.1.2.

RQ1. What are the tools that support performance testing? Our goal is to find out which quality attributes are associated with these tools, their reported strengths and limitations.

RQ2. What characterizes a performance testing tool? In order to answer this question, the following sub-questions are needed:

- **RQ2.1.** What are the elaboration approaches of the test scripts interpreted by the performance load generators?
- **RQ2.2.** What performance monitoring approaches are applied?
- **RQ2.3.** What are the persistence strategies of metrics data collected by performance monitors?

1.1.4 Search Process

Formal literature research was conducted using only databases that: (i) have a search engine capable of using keywords; and (ii) contain computer science documents. The selection includes the following bases: Association for Computing Machinery (ACM)

Digital Library[1], Engineering Village[2], IEEE Xplore[3], ScienceDirect[4], SCOPUS®[5] and SpringerLink[6]. To define the search string the terms and synonyms presented in Table 3 were used, as well as, the Boolean operator "OR" to select alternative terms and synonyms, and the Boolean operator "AND" to add more terms to the string. The resulting string can be seen in Figure 2.

Table 3 – Search string definition.

Terms	Synonyms
Performance Test	Load Test, Stress Test, Soak Test, Spike Test, Workload Test, Automation Test
Tool	Generator, Injector, Monitor, Analyzer, Framework, Suite, Environment, Plug*in
Software	Application, System

Figure 2 – Search String.

```
(("Performance Test" OR "Load Test" OR "Stress Test" OR "Spike Test" OR
"Soak Test" OR "Workload Test" OR "Automation Test") AND (Tool OR Plugin
OR Plug-In OR Framework OR Generator OR Monitor OR Injector OR Suite OR
Analyzer OR Environment) AND (Software OR System OR Application))
```

1.1.5 Inclusion and Exclusion Criteria

Inclusion Criteria

IC1. *The publication should report the use of a tool that supports performance testing.*

IC2. *The publication should propose a tool to support performance testing.*

Exclusion Criteria

EC1. *Duplicated studies.*

EC2. *The publication is not related to performance testing in the software area. e.g. performance testing of an engine.*

EC3. *The publication is written in a language other than English.*

EC4. *The publication is only available in the form of abstract, slide show, poster or short paper.*

EC5. *The publication is not available for download.*

EC6. *The publication does not report or propose a performance testing tool.*

[1] ACM: <https://www.dl.acm.org>
[2] Engineering Village: <https://www.engineeringvillage.com>
[3] IEEE: <https://www.ieeexplore.ieee.org>
[4] ScienceDirect: <https://www.sciencedirect.com>
[5] SCOPUS®: <https://www.scopus.com>
[6] SpringerLink: <https://www.link.springer.com>

1.1.6 Quality Assessment Criteria

The purpose of using quality assessment criteria is to evaluate the power from selected studies to answer some research question. The quality assessment criteria is used in two stages: the former stage being the individual evaluation of each researcher, to reduce the probability of bias; the latter stage where the researchers should reach a consensual note about the publications in a "divergent state" in the quality measurement grade.

Each of the cited QA criteria is evaluated by each researcher, according to the following degree: Yes (Y) = 1; Partial (P) = 0.5; No (N) = 0. So the total score ranging the five questions can result in: 0-1.0 (very bad); 1.5 or 2.0 (regular); 2.5 or 3.0 (good); 3.5 or 4.0 (very good); and 4.5 or 5.0 (excellent). Each of the criteria and their possible evaluations are described below:

QA1. Does the publication make a contribution to the software performance testing field?

- **Y:** A contribution is explicitly defined in the publication;
- **P:** A contribution is implied;
- **N:** No contributions could be identified.

QA2. Does the publication characterize a software performance testing tool?

- **Y:** The publication proposes and demonstrates the use of a tool;
- **P:** The publication proposes or demonstrates the use of a tool, never both;
- **N:** No, the publication does not propose or demonstrate the use of a tool.

QA3. Does the publication apply any type of empirical evaluation?

- **Y:** The publication explicitly applied an evaluation (for instance, a case study, an experiment or proof of correctness);
- **P:** The evaluation is a "toy" example;
- **N:** No evaluations could be identified.

QA4. Does the publication present some type of analysis, showing results?

- **Y:** The publication presents some type of analysis or shows the results obtained;
- **P:** The publication presents only a summary of the results;
- **N:** No form of analysis or result were presented.

QA5. Does the publication describe the techniques used in load generation and monitoring?

- **Y:** The publication explicitly describes load generation and monitoring techniques;
- **P:** The publication describes either load generation techniques, or monitoring techniques, never both;
- **N:** The publication does not describe any load generation or monitoring techniques.

1.1.7 Selection Process

The selection process is divided in five stages, which are performed by two researchers. The process steps as well as the researchers involved are described below:

(1) *Initial selection*: The search strings were generated using the selected keywords and synonyms adapting for each of the databases particularities. The initial selection encompassed all papers up to 2019 (exclusive), resulting in a total of 1673 studies. An initial selection was performed by researcher one, according to criteria EC1, EC2 and EC4 (see Section 1.1.5);

(2) *Eliminate redundancies*: at this stage, researchers one and two worked together on a pre-analysis of articles to eliminate redundancies. After the removal of duplicates, 1160 different papers remained;

(3) *Intermediate selection*: at this stage, researchers one and two read separately the title and the abstract (reading the introduction and conclusion when necessary) of each study. Here, the researchers decided to select or reject an article following IC1, IC2, EC1 - EC6 (see Section 1.1.5);

(4) *Final selection and elimination of discrepancies*: At this stage, all other studies were completely read by researchers one and two, who applied the same criteria for the intermediate selection. In case of disagreement/divergence, a third researcher would read the studies and discuss whether or not the study should be included in the final selection. This resulted in the inclusion of 146 papers;

(5) *Quality assessment*: Based on the quality criteria (see Section 1.1.6), we assessed the power of the studies to answer our research questions. The quality criteria were evaluated independently by the two researchers; therefore, reducing the probability of erroneous and/or biased results. Then researchers agreed in a consensual note on the publications that received a divergent grade. Papers that achieved at least a total score of 3 (good) and received a Yes (Y) response in QA2 were selected for data extraction. The final selection was composed of 53 papers that reported a total of 38 performance testing tools.

1.1.8 Data Extraction Strategy

To extract the relevant data from the selected publications, we produced a form that would help to answer the RQs and also to check the QA criteria. The following data were extracted for each study: title; year of publication; authors; name of the tool presented; type of license supported by the tool (commercial, academic, open-source); language or types of script supported; supported classes and types of metrics in respect to performance monitoring; reports generated on the tests performed; architecture and organization of data persistence.

When a study miss the information needed to answer all questions on the form, additional ad-hoc research was conducted. An important issue during data extraction was solved in a way that both researchers acted as data extractors and also as data verifiers, thus reducing the probability of errors and/or bias in data extraction[7]

The data presented here were manipulated using the Thoth[8] tool. This tool assisted in the whole process of this SMS, supporting the classification and extraction of data, the selection and qualification of the papers and also aided in visualizing the results.

1.1.9 Data Analysis Strategy

The data was tabulated to show: The list of published tools, its licensing and their source in Table 4 (addressing **RQ1.**); The list of published tools, supported input approaches of each and its categorization in Table 5 (addressing **RQ2.1.**); The list of published tools, their quality attributes including monitored metrics and its categorization in Table 6 (addressing **RQ2.2.**); The list of published tools, and the persistence strategies of each tool in Table 6 (addressing **RQ2.3.**).

1.2 Systematic Mapping Study Results

In this section we discuss the answers to our RQs (see Section 1.1.3). In each case, we indicate the utility of these results for researchers and practitioners.

RQ1. What are the tools that support performance testing?

The purpose of this question is to map the tools used or proposed by scientific studies that support some kind of performance testing. In total, thirty eight (38) performance testing tools were identified through our SMS. Table 4 lists these tools, their license type and the studies where they were found. Most of the tools were cited only once or twice, while some of then have been heavily referenced (11 and 9 times) showing a clear preference and greater adoption of these tools, namely LoadRunner and Apache Jmeter, the former being a commercial tool, while the latter is open source.

[7] All artifacts used in the Systematic Mapping Study are available at the Google Drive repository.
[8] Thoth: http://lesse.com.br/tools/slr

Table 4 – Tools and references.

References	Tool Name	License Type
(JOVIC et al., 2010)	Abbot	Open-Source
(Kiran; Mohapatra; Swamy, 2015; Putri; Hadi; Ramdani, 2017; AGNIHOTRI; PHALNIKAR, 2018; APTE et al., 2017; ZHANG et al., 2011; SINGH; SINGH, 2012; WU; WANG, 2010; PODELKO, 2016; KRIŽANIĆ et al., 2010)	Apache JMeter	Open-Source
(APTE et al., 2017)	AutoPerf	N/D
(ZHANG et al., 2011)	Framework CPTS	Commercial
(KIM; KIM; CHUNG, 2015; DILLENSEGER, 2009)	CLIF load injection framework	Open-Source
(Zhou; Zhou; Li, 2014)	Cloud Load Testing Framework (CLTF)	N/D
(MICHAEL et al., 2017)	CloudPerf	Commercial
(PODELKO, 2016)	CloudTest	Commercial
(CUCOS; DONCKER, 2005)	gRpas	N/D
(JOVIC et al., 2010)	Jacareto	Open-Source
(Amirante et al., 2016)	Jattack	Open-Source
(JOVIC et al., 2010)	JFCUnit	Open-Source
(DEVASENA; KUMAR; GRACE, 2017)	Load Testing Tool for Cloud (LTTC)	N/D
(ZHANG et al., 2011; Netto et al., 2011; Khan; Amjad, 2016; Chunye; Wei; Jianhua, 2017; Li; Shi; Li, 2013; YAN et al., 2011; PU; XU, 2009; Kalita; Bezboruah, 2011; PODELKO, 2016; Hamed; Kafri, 2009; RODRIGUES et al., 2014)	LoadRunner	Commercial
(PODELKO, 2016)	LoadStorm	Commercial
(JOVIC et al., 2010)	Marathon	Open-Source
(ABBORS et al., 2013)	MBPeT	Academic
(PODELKO, 2016; KRIŽANIĆ et al., 2010)	NeoLoad	Commercial
(Kim; Choi; Wong, 2009)	PJUnit	Open-Source
(RODRIGUES et al., 2015; RODRIGUES et al., 2014)	PLeTsPerf	N/D
(JOVIC et al., 2010)	Pounder	Open-Source
(FAN; MU, 2013)	Python Built-in Tool	Open-Source
(Krishnamurthy; Rolia; Majumdar, 2006)	Session-based Web Application Tester (SWAT)	N/D
(KIM; KIM; CHUNG, 2015), (PODELKO, 2016)	Silk Performer	Commercial
(BRUNE, 2017)	Simulating User Interactions (SUI)	Academic
(Kamra; Manna, 2012)	Test Harness	Commercial
(ZHANG et al., 2011; KRIŽANIĆ et al., 2010)	The Grinder	Open-Source
(RODRIGUES et al., 2014)	Visual Studio	Commercial
(KRIŽANIĆ et al., 2010; HABUL; KURTOVIC, 2008; YAN et al., 2011), (PU; XU, 2009)	WebLOAD	Commercial
(YAN et al., 2014; Yan et al., 2012a; Yan et al., 2012b)	WS-TaaS	N/D
(Maâlej; Hamza; Krichen, 2013)	WSCLT	N/D
(STUPIEC; WALKOWIAK, 2013)	Not Named Tool	N/D
(ZHANG et al., 2011; YAN et al., 2011; PU; XU, 2009)	IBM Rational Performance Tester (RPT)	Commercial
(YAN et al., 2011; PU; XU, 2009)	QALoad	Commercial
(APTE et al., 2017)	Tsung	Open-Source
(PU; XU, 2009)	Etest	N/D
(PU; XU, 2009)	OpenSTA	Open-Source
(YAN et al., 2011; PU; XU, 2009)	WAS	N/D

RQ2. What characterizes a performance testing tool?

In order to find any problems in software, the main characteristic of a performance testing tool is that it should generate a certain workload on a target system (SUT). These problems may be related to scalability, reliability, or any system bottlenecks, and this can occur in a variety of ways.

Each tool can have unique characteristics in its implementation. However, despite

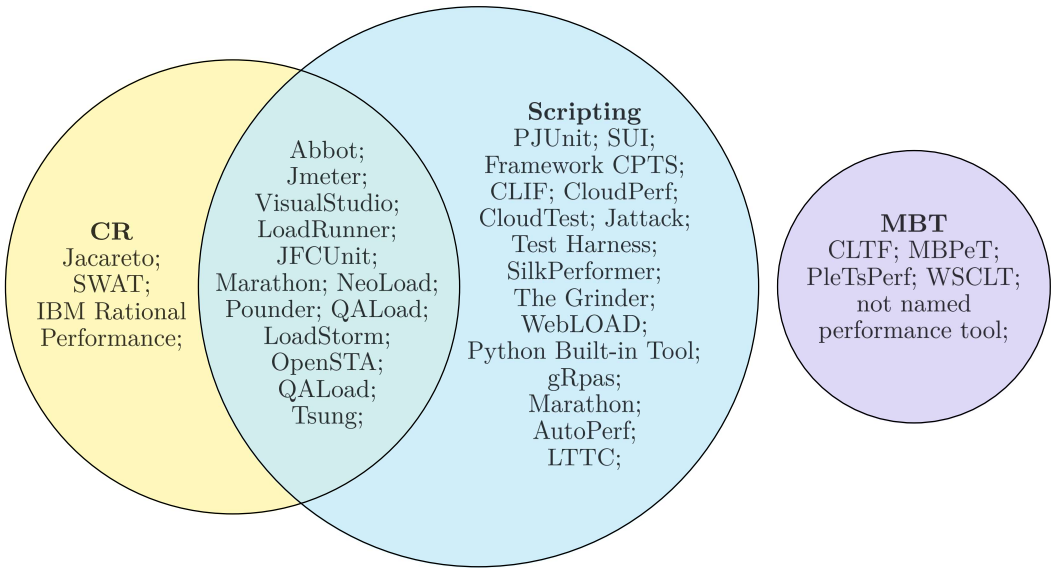
adopting distinct features and strategies, it is perceived that tools developed for this purpose make use of an already consolidated architecture.

Users of these tools may need to select a tool for a specific purpose, and selecting the most appropriate one may become a problem based on a lack of information about them. Therefore, we propose in this work a new taxonomy based on this extensive SMS, represented by feature model in Figure 4 the groups and elements of our taxonomy are presented and explained collectively in Section 1.3, including the classification of 38 tools found in the literature during the execution of this SMS. We believe this taxonomy will assist others in the process of identifying, categorizing, developing, and deploying new tools or features for performance testing and monitoring tools.

RQ2.1. What are the elaboration approaches of the test scripts interpreted by the performance load generators?

The goal of this research question is to explore different kinds of approaches for workload input definition and elaboration, and determine whether these types of input could be classified into different categories. Three main categories were observed: Model-Based Testing Dalal et al. (1999), Capture and Replay Memon e Soffa (2003), and Manual Scripting. The dispersion of tools within these categories is shown in Figure 3 and Table 5 specifies which kind of model and/or scripting language each tool supports. Choosing a tool whose model or scripting language is best known by the test engineers can result in a smaller learning curve in its use, and fewer errors when creating test scenarios. The tools that stood out most in this area were JMeter and LoadRunner, the tools were shown to support a greater number of different input types, which could explain why they were the most mentioned in **RQ1**.

Figure 3 – Input approach Venn diagram.



RQ2.2. What performance testing monitoring approaches are applied?

Performance monitoring is an ongoing process of data collection and analysis to compare how well a project, program, or policy is being implemented in relation to the expected Križanić et al. (2010). This task is fundamental in the software development life cycle and is also part of the preventive software maintenance cycle. Performance monitoring tasks are facilitated with the employment of monitoring tools. Most performance testing tools have dedicated features for monitoring, while others utilize a dedicated tool for monitoring.

Performance monitoring tools typically provide analysis of specific metrics and notifications about critical changes in the system. The selection of an appropriate tool for monitoring should be given in relation to which metrics one wishes to collect to analyze the performance requirements of the application being tested.

To answer this research question in detail, the data were classified according to the monitoring approaches found in the primary studies resulting from this SMS and in accordance with the metrics selected in Section 1.3.3.1. The first approach refers to metrics directly related to the application, such as: response time, hits per second, responses per second, transactions per second, transaction success rate, number of virtual users, and total test time. The second approach presents metrics related to the resources from which the system under test (SUT) is hosted, which are classified as CPU, memory, I/O and network utilization.

All tools analyzed use metrics of the two approaches represented. In general, the tools monitored the SUT metrics more than the application itself. The reason is that, in the application metrics approach, the data obtained during and after the tests execution needed to be interpreted to be shown in a clear and objective way to the user. Meanwhile, the metrics related to the SUT are only data captured at certain moments in the workload execution and shown to the user.

The results obtained during this classification were represented in Table 6, where it is possible to visualize each monitoring metric that the tool in question has.

RQ2.3. What are the persistence strategies of metrics data collected by performance testing monitors?

To evaluate the performance of a system, it is necessary to monitor its behavior during workload execution. This results in a high-volume of data persisted for later analysis, thus making the implementation of a persistence layer necessary. As consequence, most persistence layers will use external files for persistence or underlying database management system. So the file types that have been observed as most common are XML and JSON, respectively (see Table 6). Finally, as for database management systems, no patterns or preferences were identified.

Table 5 – Tools and workload input approaches.

Tool Name	CR	MBT					Scripting Language															
	Capture Replay	PTA	SAM	SWM	UML AD	UML UC	.Net	BeanShell	C	C#	C++	Clojure	Groovy	Java	JavaScript	JSON	Jython	Python	Ruby	Scala	SCL	XML
Abbot	✓													✓								
Apache JMeter	✓							✓					✓		✓					✓		
AutoPerf																						✓
CLIF																						✓
CLTF			✓																			
CloudPerf																						✓
CloudTest															✓							
gRpas																						✓
IBM RPT	✓																					
Jacareto	✓																					
Jattack															✓							
JFCUnit	✓													✓								✓
LTTC														✓								
LoadRunner	✓						✓		✓					✓	✓							
LoadStorm	✓															✓						✓
Marathon	✓																		✓			
MBPeT		✓																				
NeoLoad	✓														✓							
Not Named Tool				✓																		
OpenSTA	✓																				✓	
PJUnit														✓								
PLeTsPerf					✓	✓																
Pounder	✓													✓								
Python Built-in Tool																		✓				
QALoad	✓										✓			✓								
SWAT	✓																					
Silk Performer										✓												
SUI Test														✓								
Harness The Grinder														✓				✓				
Tsung	✓											✓					✓					
Visual Studio	✓								✓													✓
WebLOAD											✓											
WSCLT		✓																				
Etest*																						
Framework																						
CPTS*																						
WAS*																						
WS-TaaS*																						

∗It was not possible to find information on the input approach of these tools in the literature or by "ad-hoc" manner.
PTA: Probabilistic Timed Automata; **SAM**: Sequencial Action Model; **SWM**: Stochastic Workload Model; **UML AD**: UML Activity Diagram; **UML UC**: UML Use Case diagram.

Table 6 – Monitored metrics and Persistence strategies.

Tool Name	Monitored Metrics: Application: Response time	Hits per sec.	Responses per second	Transactions per second	Transaction success rate	#Virtual Users	Total test time	SUT Resources: CPU	Memory	I/O	Network	Persistence strategy: Proprietary files	Databases	Html	JSON	Plain Text	JDBC Drivers	XML
Abbot	✓							✓	✓									
Apache JMeter	✓	✓	✓	✓	✓	✓	✓	✓			✓				✓		✓	✓
AutoPerf	✓					✓		✓			✓					✓		
CLIF load injection framework	✓				✓			✓	✓		✓						✓	✓
Cloud Load Testing Framework (CLTF)	✓																	
CloudPerf	✓							✓	✓	✓	✓		✓					
CloudTest	✓	✓		✓		✓	✓	✓		✓					✓			
Framework CPTS	✓	✓	✓	✓	✓	✓		✓			✓		✓		✓			
gRpas	✓						✓	✓		✓								✓
IBM Rational Performance Tester	✓	✓	✓		✓			✓	✓	✓	✓		✓					
Jacareto	✓							✓	✓									✓
Jattack	✓				✓			✓							✓			✓
JFCUnit	✓							✓	✓				✓		✓			✓
Load Testing Tool for Cloud (LTTC)	✓					✓		✓	✓	✓	✓							
LoadRunner	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓		
LoadStorm	✓	✓			✓		✓	✓	✓	✓	✓							✓
Marathon	✓							✓	✓									
MBPeT	✓				✓	✓		✓	✓	✓	✓							
NeoLoad	✓			✓		✓		✓	✓		✓				✓			
Not Named Tool	✓							✓										
OpenSTA	✓						✓	✓										
PJUnit	✓										✓				✓			✓
Pounder	✓							✓	✓						✓			✓
Python Built-in Tool		✓					✓	✓	✓		✓							
QALoad						✓	✓					✓						
Session-based Web Application Tester (SWAT)	✓							✓		✓	✓							
Silk Performer	✓				✓	✓	✓	✓	✓		✓							
Simulating User Interactions (SUI)	✓							✓	✓	✓	✓				✓			✓
Test Harness	✓							✓			✓				✓			
The Grinder	✓			✓				✓	✓	✓	✓							
Tsung	✓			✓		✓		✓		✓	✓							✓
Microsoft Visual Studio	✓			✓	✓		✓	✓	✓	✓	✓	✓	✓					✓
WAS	✓							✓			✓							
WebLOAD	✓					✓		✓	✓	✓	✓				✓	✓		✓
WS-TaaS	✓	✓		✓	✓	✓	✓	✓	✓	✓				✓				
WSCLT	✓			✓			✓	✓			✓							
PLeTsPerf*																		
Etest*																		

∗ It was not possible to find information on the monitored metrics or persistence strategies of these tools in the literature or by "ad-hoc" manner.

1.3 Taxonomy of Performance Testing Tools

A taxonomy is a scientific method of classification according to an established system in a specific domain, with the resulting catalog used to provide a framework for analysis. Any taxonomy should take into account the importance of separating elements of a group into subgroups that are unambiguous, and taken together include all possibilities (CLARKE; MALLOY, 2001).

The main objective of our taxonomy is to reduce the gap between practice and research in performance testing tools, especially when it comes to the terms used and the approaches implemented in each one. This taxonomy provides means of comparison and evaluation of the tools features that can be useful in deciding which tool to use or how to design future systematic mapping. Not all features of the tools represented in the taxonomy were planned to be identified in our initial research research perspective, but are nevertheless identified and are represented in the taxonomy.

1.3.1 Input Approaches

Refers to the input approach that the tools support and/or provide means of elaboration. They were divided as follows:

1.3.1.1 Capture Replay

Capture Replay (CR), also known as Record and Playback, is a technique where a test engineer performs the tests manually once in the application. This is, by interacting with the graphical user interface (GUI) in "capture" mode, the tool stores this interaction and outputs a test script that can be "replayed" by the tool multiple times by several Virtual Users (VU). The continuous modification of a GUI may render these types of tests obsolete, forcing the test engineers to re-capture those tests. However, modern CR tools do not rely solely on coordinates for test case execution but maintain extra information such as the handle, type, and label (if any) of the elements, enabling the replayer to locate the element when it has been moved (MEMON; SOFFA, 2003). Sometimes, this technique also employees manual script editing for the removal of random generated values, hard coded values and enhancements whenever possible.

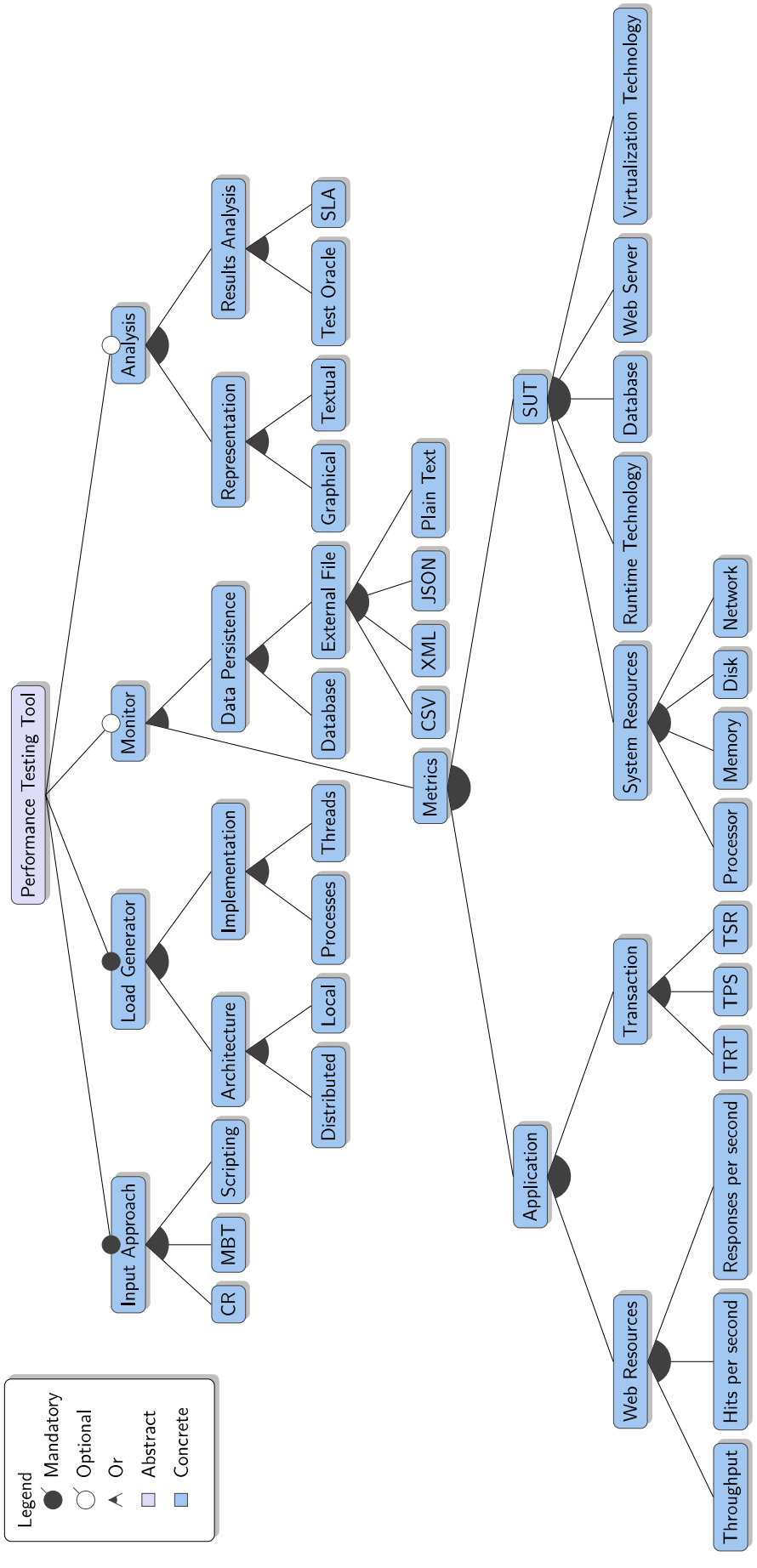


Figure 4 – Taxonomy of performance testing tools represented by feature model

Source: the author.

1.3.1.2 Model-Based Testing

Model-Based Testing (MBT) approach involves developing and using a data model to generate tests. The model is essentially a specification of the inputs to the software, and can be developed early in the cycle from requirements information. It can be especially effective for systems that are changed frequently, because testers can update the data model and then rapidly regenerate a test suite, avoiding tedious and error-prone editing of a suite of hand-crafted tests Dalal et al. (1999) or even tests that were created using the CR technique. In our research we were able to find tools using as input Probabilistic Timed Automata (PTA), Sequential Action Models (SAM), Stochastic Workload Models, UML Activity Diagrams, and UML Use Case Diagrams.

1.3.1.3 Scripting

Manual script writing technique in which the test engineer manually writes a set of code statements, into a defined programming language, that will be executed by the load generator in the form of Virtual Users (VU).

1.3.2 Load Generator

This group represents the often called "module" of load generation. Its the core of many performance testing tools, it is responsible for interpreting the scripts and generating the correspondent workload in the SUT. It employs the creation and management of multiple VU, which can be executed locally or in a distributed manner, utilizing a master/slave approach.

1.3.2.1 Architecture

The architecture of a load generator deals mainly with the organization of its elements, which can be organized in a: (i) **Local** architecture when the VU are created and run in a single machine. This severely impacts the quality of the results obtained through performance testing, since they rely on the amount of workload that can be generated and maintained in a SUT. Limiting the load generation to one machine only, however broad it may be, limits the amount of load that can be generated, creating a bottleneck in the load generator itself; (ii) **Distributed** architecture load generators on the other hand, as the name implies, distribute the load of generating VU in a master/slave manner. This architecture enables having a local master controller that handles the test distribution and execution on the slaves, which are remote instances that will send the requests to the SUT. This architecture adds another layer of complexity onto load generators, as the test engineers will have to set up multiple computers and/or utilize cloud services, such as Azure, AWS or Google Cloud.

Another difficulty when utilizing distributed architectures, is how to handle parameterized data in tests. This is because that first, if the load generator master controller does not handle the distribution of parameters, you will need to have separated files, and second, if the test engineer wants to update, them he has to go through each slave node to make the modifications.

1.3.2.2 Implementation

Characterizes the low level representation for load generators implementations. We were able to identify, albeit not in all cases, two different implementation approaches: (i) creating different **process** for each instance of a Virtual Users (VU), which do not share the same memory space, and are independent to each other. This is important for VU isolation, so that a problem within an instance of a VU does not affect the rest; (ii) The use of multiple **threads** for the VU execution, which shares the same memory address, lowering the communication cost between VUs. On the other hand, a problem within a VU will certainly affect the others and the reliability of the load generator itself.

1.3.3 Monitor

Refers to the monitoring modules present in some tools, the metrics they monitor as well as the data persistence approaches taken.

1.3.3.1 Metrics

A software metric is a measure of software characteristics which are quantifiable or countable. Software metrics are important for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses. Metrics capture a value pertaining to systems at a specific point in time, like the number of users currently logged in to a web application or CPU usage. Therefore, metrics are usually collected once per second, per minute, or at another regular interval to monitor a system over time.

There are two important subcategories of metrics in our taxonomy:

(1) Application Metrics: indicates the top-level health of the system by measuring its useful output.

(i) **Web Resources**: These are vital performance counters for assessment of Web application ability to maintain the workload simulated. (a) *Throughput*: Shows the amount of server throughput during each second of the load test scenario run. Throughput measures the actual rate at which work requests are completed; (b) *Hits Per Second*: Shows the number of requests per second; (c) *Responses Per Second*: shows the number of HTTP status codes returned from the Web server during each second of the load test

scenario run, grouped by status code.

(ii) **Transaction**: During load test scenario execution, VU generate data as they perform transactions. This metric enables collecting data that shows the transaction performance and status throughout script execution, in which are presented as follows: (a) *Transaction Response Time (TRT)*: Different response time values under different load. Average response time, maximum, percentile, and so on; (b) *Transaction Per Second (TPS)*: Shows the number of transactions generated per second; (c) *Transaction Success Rate (TSR)*: Shows the number of transactions that passed, failed, or stopped.

(2) System Under Test (SUT) Metrics: Most components of software infrastructure serve as a resource for monitoring systems. ***System Resources:*** Some resources are low-level, *e.g.* a server's resources include such physical components as processor, memory, disks, and network. Each one of them have a list of performance counters that could be used to measure the performance requirements of SUT, such as:

(i) **Processor**: Program execution threads consume processor (CPU) resources. Available performance counters measure how much CPU processing time threads and other executable units of work consume. These processor utilization measurements allow to determine which applications are responsible for CPU consumption. The processor performance counter are presented as follows: (a) *% Processor Time*; (b) *% Interrupt Time*; (c) *Processor Queue Length.*

(ii) **Memory**: A shortage of RAM is often evident indirectly as a disk performance problem, when excessive paging to disk consumes too much of the available disk bandwidth. Consequently, paging rates to disk are an important memory performance indicator. When observing a shortage of available RAM, it is often important to determine how the allocated physical memory is being used and count resident pages of a problematic process known as its working set. Instances of memory performance counters are shown as follows: (a) *Available Bytes*; (b) *Working Set*; (c) *Page Reads/Sec.*

(iii) **Disk**: Through the I/O manager stack, an operation system maintains physical and logical disk operations. A physical disk is the internal representation of specific storage device.It is important to be proactive about disk performance because it tends to degrade rapidly, particularly when disk-paging activity occurs. Examples of disk performance counters are presented, such as: (a) *Avg. Disk secs/transfer*; (b) *% Idle Time*; (c) *Disk Transfers/Sec*; (d) *Avg. Disk Queue Length.*

(iv) **Network**: Networking performance has become ever more important today with the proliferation of distributed and cloud applications. Network interface statistics are gathered by software embedded in the network interface driver layer. This software counts the number of packets that are sent and received. Networking bottlenecks are tricky to catch and analyze. Packet rates, collision rates and error rates do not always point to the cause of the problem. (a) *Bytes Total/Sec*; (b) *Server Bytes Total/Sec*; (c) *Connections Established.*

Runtime Technology: Application performance also depends on the architectural level monitoring and tuning. However, architectural design is built upon specific technologies. Each platform differs in which metrics and counters impact on the application performance. Common examples of runtime technologies are Java Platform Enterprise Edition (Java EE) or the .NET Framework.

Database: It is imperative to ensure optimal performance of the database as this is essential to any data-driven application. There are many factors affecting overall application performance that may come from the database side, such as: Poor database design; Poor logic used in queries; Database server machines dedicated to multiple applications.

Web Server: The function of a Web server is to service requests made through the HTTP protocol. Some Web servers even provide modules presenting information on server activity for automating the monitoring process.

Virtualization Technology: Virtualization platforms provide the service of creating a virtual (software) version of hardware. This adds another layer to complexity and computational efforts which also needs to be monitored and tuned for better results performance wise. Virtualization technology metrics can be very similar to those of System Resources, depending on the virtualization platform.

1.3.3.2 Data Persistence

The most common approaches for storing the data results from monitoring are the use of external files (like CSV, XML, JSON or even as plain text) or databases systems, for instance SQL or NoSQL.

1.3.4 Analysis

Refers to how the data results from the monitoring is processed and represented in performance testing reports.

1.3.4.1 Representation

How results are being presented to the test engineer, it could be a graphical and/or textual data representation using different techniques to generate a performance testing report. For instance, Word, PDF or HTML documents.

1.3.4.2 Result Analysis

What techniques, methods or approaches of automatic data analysis the tool applies in the measured data results. For instance, a test oracle or a Service Level Agreement (SLA) (LEE; BEN-NATAN, 2002).

1.4 Threats to Validity

In this section, the threats identified in the context of this study are described as suggested by (COOK; CAMPBELL, 1979).

Construct Validity: This is a threat that affect the statements in this paper: provide an empirical reference that serves as a starting point decision making in selecting testing performance tools. In this sense, it is important to reassert that our analysis is built on well accepted guidelines for performing SMS in SE proposed by (Engström; Petersen, 2015), including a research.

First of all, systematic mappings are known for not guaranteeing the inclusion of all the relevant works on the field. This can be explained by the limitation of the search mechanisms for set of keywords defined in this study and the lack of them in some of the relevant works. In order to avoid this bias, articles not found in database where manually inserted into the body of knowledge.

Internal Validity: This type of threats is related to how we ensure that the performed analysis is valid to the problem statement. Likewise, in order to reduce possible bias, the stages of selection of the studies and data extraction were carried out by two researchers. The results found by each were tabulated and compared, so that any kind of bias could be identified, and when in disagreement, the authors could debate and a consensus was reached.

External Validity: The use of a well-defined and validated protocol assures that any other group of researchers could replicate this mapping using the same set of parameters would yield the same results. The only variable that could compromise this assumption is time, as new researches and tools emerge everyday. To minimize this threat the update of this SMS in the future is required.

Conclusion Validity: This research found a relatively good number of focused papers, thus providing a statistical power to drive our conclusions. This could be affected by terminological problems in the search string, which may have led to the absence of some primary studies. For the minimization of these problems, the generated string was tested and the results were previously analyzed in a way that one could notice the relevance of the same. When necessary, the search string was modified and the process was redone. Finally, we reduced the threat of not indexing all available content on the web by using six (6) digital libraries.

1.5 Related Work - Systematic Mapping Review

This section summarizes main contributions from related works shown in Table 7.

The survey conducted by Jiang e Hassan, summarizes various test type definitions (Load Testing, Performance Testing and Stress Testing). In addition, specifies the relationship between them and verifies the techniques that are used in the three phases of

performance testing: the workload design phase, the load execution phase and the performance testing analysis phase. For each of these phases, some open research issues are provided.

To clarify concepts, objectives and types of performance testing, Sharmila presents a brief description of nine (9) performance testing tools found in the market through an empirical study. In this study, the authors list the main characteristics that performance tests aim to identify in a given system.

In order to deepen the search for performance testing tools, Isha conducted a research on performance testing tools for web applications. In this study, eighteen (18) performance testing tools were found, presenting their important characteristics.

From this mapping, the authors constructed a taxonomy on software testing, that captured both the perspective of the problem and solution. The authors reinforce the idea that a taxonomy should be expandable, so that the initial structure should be assessed as sound and beneficial from researchers and practitioners perspectives.

Although the articles discussed in this section cover several aspects regarding performance testing tools, they fail to systematically investigate the available tools and their characteristics. Therefore, it is relevant to review which performance testing tools are available or are applied and which approaches and techniques are employed. A broad view of these tools would allow a greater understanding of the peculiarities of each and also an empirically supported background in decision making as benefit to the research and practice.

Table 7 – Related work summary.

Concept	**Our Study**	**Sharmila and Ramadevi** (Sharmila, 2014)	**Isha and Vikram** (Isha, 2015)	**Engstrom and Petersen** (Engström; Petersen, 2015)	**Jiang and Hassan** (Jiang; Hassan, 2015)
Paper Type	SLR	Ad-hoc review	Survey	Proposal	Survey
Interval	up to Dec 2018	not specified	2000 - 2014	not specified	1993 - 2013
Contribution	overview of tools and tools taxonomy	overview of tools	overview of tools	testing taxonomy	compares the state of research and practice
# Tools	38	9	18	0	not specified

1.6 Chapter Summary

This chapter presented an SMS on performance testing tools. The developed protocol and it's execution was described in Section 1.1, its results, the tools found, and the answers to the research questions are discussed in Section 1.2. The main contribution of this mapping is described in Section 1.3 in the form of a taxonomy of performance testing tools.

Bibliography

ABBORS, F. et al. Model-based performance testing in the cloud using the MBPeT tool. In: . [S.l.: s.n.], 2013. p. 423–424. Cited in page 27.

AGNIHOTRI, J.; PHALNIKAR, R. Development of Performance Testing Suite Using Apache JMeter. In: BHALLA, S. et al. (Ed.). **Proc. Intelligent Computing and Information and Communication**. Singapore: Springer Singapore, 2018. p. 317–326. ISBN 978-981-10-7245-1. Cited in page 27.

Amirante, A. et al. Jattack: a WebRTC load testing tool. In: **Proc. Principles, Systems and Applications of IP Telecommunications**. [S.l.: s.n.], 2016. p. 1–6. Cited in page 27.

APTE, V. et al. AutoPerf: Automated Load Testing and Resource Usage Profiling of Multi-Tier Internet Applications. In: **Proc. ACM/SPEC International Conference on Performance Engineering**. New York, NY, USA: [s.n.], 2017. (ICPE '17), p. 115–126. ISBN 978-1-4503-4404-3. Disponível em: <http://doi.acm.org/10.1145/3030207.3030222>. Cited in page 27.

BARN, B.; BARAT, S.; CLARK, T. Conducting systematic literature reviews and systematic mapping studies. In: **Proceedings of the 10th Innovations in Software Engineering Conference**. New York, NY, USA: ACM, 2017. (ISEC '17), p. 212–213. ISBN 978-1-4503-4856-0. Disponível em: <http://doi.acm.org/10.1145/3021460.3021489>. Cited in page 21.

BRUNE, P. Simulating User Interactions: A Model and Tool for Semi-realistic Load Testing of Social App Backend Web Services. In: **Proc. WEBIST**. [S.l.: s.n.], 2017. p. 235–242. Cited in page 27.

Chunye, D.; Wei, S.; Jianhua, W. Based on the analysis of mobile terminal application software performance test. In: **Proc. IEEE/ACIS 18th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing**. [S.l.: s.n.], 2017. p. 391–395. Cited in page 27.

CLARKE, P.; MALLOY, B. A. A Unified Approach to Implementation-Based Testing of Classes. In: **Proc. 1st Annual International Conference on Computer and Information Science**. [S.l.: s.n.], 2001. Cited in page 32.

COOK, T.; CAMPBELL, D. **Quasi-Experimentation: Design and Analysis Issues for Field Settings**. [S.l.]: Houghton Mifflin, 1979. Cited in page 38.

CUCOS, L.; DONCKER, E. de. "gRpas", a Tool for Performance Testing and Analysis. In: **Proc. Springer-Verlag 5th International Conference on Computational Science - Volume Part I**. Berlin, Heidelberg: [s.n.], 2005. (ICCS'05), p. 322–329. ISBN 3-540-26032-3, 978-3-540-26032-5. Disponível em: <https://doi.org/10.1007/11428831_40>. Cited in page 27.

Dalal, S. R. et al. Model-based testing in practice. In: **Proc. IEEE International Conference on Software Engineering**. [S.l.: s.n.], 1999. p. 285–294. ISSN 0270-5257. Cited 2 times in pages 28 and 34.

DEVASENA, M. S. G.; KUMAR, V. K.; GRACE, R. K. LTTC: A Load Testing Tool for Cloud. In: MODI, N.; VERMA, P.; TRIVEDI, B. (Ed.). **Proc. Springer Singapore International Conference on Communication and Networks**. [S.l.: s.n.], 2017. p. 689–698. ISBN 978-981-10-2750-5. Cited in page 27.

DILLENSEGER, B. CLIF, a framework based on Fractal for flexible, distributed load testing. **annals of telecommunications - annales des télécommunications**, v. 64, n. 1, p. 101–120, 2009. ISSN 1958-9395. Disponível em: <https://doi.org/10.1007/s12243-008-0067-9>. Cited in page 27.

Engström, E.; Petersen, K. Mapping software testing practice with software testing research — SERP-test taxonomy. In: **Proc. IEEE 8th International Conference on Software Testing, Verification and Validation Workshops**. [S.l.: s.n.], 2015. p. 1–4. Cited 3 times in pages 21, 38, and 39.

FAN, H.; MU, Y. A performance testing and optimization tool for system developed by Python language. Institution of Engineering and Technology, p. 24–27(3), 2013. Cited in page 27.

HABUL, A.; KURTOVIC, E. Load testing an AJAX application. In: **Proc. IEEE 30th International Conference on Information Technology Interfaces**. [S.l.: s.n.], 2008. p. 729–732. Cited in page 27.

Hamed, O.; Kafri, N. Performance testing for web based application architectures (.NET vs. Java EE). In: **Proc. First International Conference on Networked Digital Technologies**. [S.l.: s.n.], 2009. p. 218–224. ISSN 2155-8728. Cited in page 27.

Isha, B. V. A. A Brief Survey on Web Application Performance Testing Tools Literature Review. In: INTERNATIONAL JOURNAL OF LATEST TRENDS IN ENGINEERING AND TECHNOLOGY. [S.l.], 2015. Cited in page 39.

Jiang, Z. M.; Hassan, A. E. A Survey on Load Testing of Large-Scale Software Systems. **IEEE Transactions on Software Engineering**, v. 41, n. 11, p. 1091–1118, 2015. ISSN 0098-5589. Cited 2 times in pages 38 and 39.

JOVIC, M. et al. Automating Performance Testing of Interactive Java Applications. In: **Proc. ACM 5th Workshop on Automation of Software Test**. New York, NY, USA: [s.n.], 2010. (AST '10), p. 8–15. ISBN 978-1-60558-970-1. Cited in page 27.

Kalita, M.; Bezboruah, T. Investigation on performance testing and evaluation of PReWebD: a .NET technique for implementing web application. **IET Software**, v. 5, n. 4, p. 357–365, 2011. ISSN 1751-8806. Cited in page 27.

Kamra, M.; Manna, R. Performance of Cloud-Based Scalability and Load with an Automation Testing Tool in Virtual World. In: **Proc. IEEE 8th World Congress on Services**. [S.l.: s.n.], 2012. p. 57–64. ISSN 2378-3818. Cited in page 27.

Khan, R.; Amjad, M. Web application's performance testing using HP LoadRunner and CA Wily introscope tools. In: **Proc. International Conference on Computing, Communication and Automation**. [S.l.: s.n.], 2016. p. 802–806. Cited in page 27.

KIM, G.-H.; KIM, Y.-G.; CHUNG, K.-Y. Towards virtualized and automated software performance test architecture. **Multimedia Tools and Applications**, v. 74, n. 20, p. 8745–8759, 2015. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-013-1536-3>. Cited in page 27.

Kim, H.; Choi, B.; Wong, W. E. Performance Testing of Mobile Applications at the Unit Test Level. In: **Proc. IEEE 3rd International Conference on Secure Software Integration and Reliability Improvement**. [S.l.: s.n.], 2009. p. 171–180. Cited in page 27.

Kiran, S.; Mohapatra, A.; Swamy, R. Experiences in performance testing of web applications with Unified Authentication platform using Jmeter. In: **Proc. International Symposium on Technology Management and Emerging Technologies**. [S.l.: s.n.], 2015. p. 74–78. Cited in page 27.

KITCHENHAM, B. A. **Guidelines for performing Systematic Literature Reviews in software engineering. EBSE Technical Report EBSE-2007-01**. [S.l.: s.n.], 2007. Cited in page 22.

KOZIOLEK, H. Goal, question, metric. In: **Dependability metrics**. [S.l.]: Springer, 2008. p. 39–42. Cited in page 21.

Krishnamurthy, D.; Rolia, J. A.; Majumdar, S. A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems. **IEEE Transactions on Software Engineering**, v. 32, n. 11, p. 868–882, 2006. ISSN 0098-5589. Cited in page 27.

KRIŽANIĆ, J. et al. Load testing and performance monitoring tools in use with AJAX based web applications. In: **Proc. IEEE 33rd International Convention MIPRO**. [S.l.: s.n.], 2010. p. 428–434. Cited in page 27.

Križanić, J. et al. Load testing and performance monitoring tools in use with AJAX based web applications. In: **33rd International Convention MIPRO**. [S.l.: s.n.], 2010. p. 428–434. Cited in page 29.

LEE, J.; BEN-NATAN, R. **Integrating Service Level Agreements: Optimizing Your OSS for SLA Delivery**. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 0471428663. Cited in page 37.

Li, P.; Shi, D.; Li, J. Performance test and bottle analysis based on scientific research management platform. In: **Proc. 10th International Computer Conference on Wavelet Active Media Technology and Information Processing**. [S.l.: s.n.], 2013. p. 218–221. Cited in page 27.

Maâlej, A. J.; Hamza, M.; Krichen, M. WSCLT: A Tool for WS-BPEL Compositions Load Testing. In: **Proc. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises**. [S.l.: s.n.], 2013. p. 272–277. ISSN 1524-4547. Cited in page 27.

MEMON, A. M.; SOFFA, M. L. Regression testing of GUIs. **ACM SIGSOFT Software Engineering Notes**, ACM, v. 28, n. 5, p. 118–127, 2003. Cited 2 times in pages 28 and 32.

MICHAEL, N. et al. CloudPerf: A Performance Test Framework for Distributed and Dynamic Multi-Tenant Environments. In: **Proc. ACM/SPEC 8th International Conference on Performance Engineering**. New York, NY, USA: ACM, 2017. (ICPE '17), p. 189–200. ISBN 978-1-4503-4404-3. Disponível em: <http://doi.acm.org/10.1145/3030207.3044530>. Cited in page 27.

Netto, M. A. S. et al. Evaluating Load Generation in Virtualized Environments for Software Performance Testing. In: **Proc. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum**. [S.l.: s.n.], 2011. p. 993–1000. ISSN 1530-2075. Cited in page 27.

PODELKO, A. Reinventing performance testing. In: CMG. [S.l.], 2016. Cited in page 27.

PU, Y.; XU, M. Load testing for web applications. In: **Proc. IEEE First International Conference on Information Science and Engineering**. [S.l.: s.n.], 2009. p. 2954–2957. Cited in page 27.

Putri, M. A.; Hadi, H. N.; Ramdani, F. Performance testing analysis on web application: Study case student admission web system. In: **Proc. International Conference on Sustainable Information Engineering and Technology**. [S.l.: s.n.], 2017. p. 1–5. Cited in page 27.

RODRIGUES, E. M. et al. PLeTsPerf - A Model-Based Performance Testing Tool. In: **Proc. IEEE 8th International Conference on Software Testing, Verification and Validation**. [S.l.: s.n.], 2015. p. 1–8. ISSN 2159-4848. Cited in page 27.

RODRIGUES, E. M. et al. Evaluating capture and replay and model-based performance testing tools: an empirical comparison. In: **Proc. ACM 8th International Symposium on Empirical Software Engineering and Measurement**. [S.l.: s.n.], 2014. p. 9. Cited in page 27.

Sharmila, E. R. S. Analysis of Performance Testing on Web Applications. In: INTERNATIONAL JOURNAL OF ADVANCED RESEARCH IN COMPUTER AND COMMUNICATION ENGINEERING. **International Journal of Advanced Research in Computer and Communication Engineering**. [S.l.], 2014. Cited in page 39.

SINGH, M.; SINGH, R. Load Testing of web frameworks. In: . [S.l.: s.n.], 2012. p. 592–596. ISBN 978-1-4673-2922-4. Cited in page 27.

STUPIEC, E.; WALKOWIAK, T. Automatic Load Testing of Web Application in SaaS Model. In: ZAMOJSKI, W. et al. (Ed.). **Proc. Springer International New Results in Dependability and Computer Systems**. Heidelberg: [s.n.], 2013. p. 421–430. ISBN 978-3-319-00945-2. Cited in page 27.

WU, Q.; WANG, Y. Performance testing and optimization of J2EE-based web applications. In: **Proc. IEEE Second International Workshop on Education Technology and Computer Science**. [S.l.: s.n.], 2010. v. 2, p. 681–683. Cited in page 27.

YAN, M. et al. Delivering Web service load testing as a service with a global cloud. v. 27, n. 3, 2014. Cited in page 27.

Yan, M. et al. Building a TaaS Platform for Web Service Load Testing. In: **Proc. IEEE International Conference on Cluster Computing**. [S.l.: s.n.], 2012. p. 576–579. ISSN 1552-5244. Cited in page 27.

Yan, M. et al. WS-TaaS: A Testing as a Service Platform for Web Service Load Testing. In: **Proc. IEEE 18th International Conference on Parallel and Distributed Systems**. [S.l.: s.n.], 2012. p. 456–463. ISSN 1521-9097. Cited in page 27.

YAN, X. et al. Performance Testing of Open Laboratory Management System Based on LoadRunner. In: **Proc. IEEE First International Conference on Instrumentation, Measurement, Computer, Communication and Control**. [S.l.: s.n.], 2011. p. 164–167. Cited in page 27.

ZHANG, L. et al. Design and implementation of cloud-based performance testing system for web services. In: **Proc. IEEE 6th International Conference on Communications and Networking in China**. [S.l.: s.n.], 2011. p. 875–880. Cited in page 27.

Zhou, J.; Zhou, B.; Li, S. Automated Model-Based Performance Testing for PaaS Cloud Services. In: **Proc. IEEE 38th International Computer Software and Applications Conference Workshops**. [S.l.: s.n.], 2014. p. 644–649. Cited in page 27.

Index