

FEDERAL UNIVERSITY OF PAMPA

Gustavo Girardon dos Reis

**PerfMoon: Proposal of a Tool for
Monitoring the Performance of Web
Applications**

Alegrete
2019

Gustavo Girardon dos Reis

PerfMoon: Proposal of a Tool for Monitoring the Performance of Web Applications

Term Paper presented in Software Engineering Graduation Course in the Federal University of Pampa as a partial requirement for obtaining the title of Software Engineering Bachelor

Supervisor: Maicon Bernardino da Silveira

Co-supervisor: Elder de Macedo Rodrigues

Alegrete
2019

Gustavo Girardon dos Reis

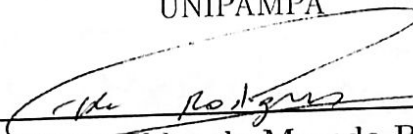
PerfMoon: Proposal of a Tool for Monitoring the Performance of Web Applications

Term Paper presented in Software Engineering Graduation Course in the Federal University of Pampa as a partial requirement for obtaining the title of Software Engineering Bachelor

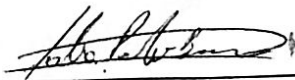
Term Paper presented and approved on 26 NOV of 2013
Committee members:



Prof. PhD. Maicon Bernardino da Silveira
Supervisor
UNIPAMPA



Prof. PhD. Elder de Macedo Rodrigues
Co-supervisor
UNIPAMPA



Prof. PhD. Fábio Paulo Basco
UNIPAMPA



Prof. Dr. Avelina Francisco Zorzo
PUCRS

Acknowledgements

First of all, I would like to thank my parents Sueli and Juca. They always supported unrestrictedly my decisions, and endowed me with moral values. I also want to thank my love, Évelen, that held the pieces together. Be sure, my love, that it is reciprocal, and being with you is of major importance to me. Having you by my side gave me strength and inspiration to overcome the challenges I have been facing.

Probably I would never have been able to finish this term paper without the guidance of my advisor, help from friends, girlfriend and support from my family.

I would like to express my deepest gratitude to my advisor and friend, Maicon Bernardino, for his guidance and patience.

Thanks to my colleagues from the LESSE Group at UNIPAMPA, Victor Costa, Rodrigo Machado, Neto Iung and Allan Pedroso who were always willing to help and give the best suggestions. It would have been a lonely journey without you.

Thank you to all people who have helped me in any way in this journey and who I did not mention here.

Thank you all!

*“I have not failed.
I’ve just found 10,000 ways that won’t work.”
(Thomas A. Edison)*

Abstract

The knowledge and application of tools to automate performance testing and monitoring is essential to ensure software reliability and therefore its quality. To investigate, identify and characterize existing performance testing tools reported in the literature, a protocol was formulated and executed according to the guidelines for performing systematic mapping in Software Engineering. The performance testing and monitoring tools were classified according to their relevance in the literature, highlighting the most commonly used tools, their supported input approaches, workload approaches, monitored metrics and logging strategies. From the analysis of these results a taxonomy on performance monitoring tools was proposed. With the results of this study, it was possible to quantify and qualify research related to existing performance testing tools in the literature, and also to characterize these tools. This study also proposes a Performance Monitoring tool called PerfMoon, which is a versatile and light-weight, also collects hundreds of thousands of metrics per second and delivers this information as time-series in near real-time. Data is collected using methods which collect and periodically send their metrics data to another analysis tool. Designed as a component that is integrated into a larger system, PerfMoon is light-weight, with little impact on the system resources and it is easy to install and to configure. This study also contemplates an experimental benchmark where the developed tool has its results compared with another tool.

Key-words: Monitoring, Application software, Software quality, Costs, Software testing, Performance monitoring, Performance analysis

Resumo

O conhecimento e a aplicação de ferramentas para automatizar testes de desempenho e monitoramento é uma tarefa essencial para garantir a confiabilidade do software e, portanto, sua qualidade. Para investigar, identificar e caracterizar ferramentas de testes de desempenho existentes relatadas na literatura, um protocolo foi formulado e executado de acordo com as diretrizes para a realização de mapeamentos sistemáticos em Engenharia de Software. As ferramentas de teste e monitoramento de desempenho foram classificadas de acordo com sua relevância na literatura, destacando as ferramentas mais comumente usadas, suas abordagens de entrada suportadas, abordagens de carga de trabalho, métricas monitoradas e estratégias de registro. A partir da análise desses resultados, foi proposta uma taxonomia sobre ferramentas de monitoramento de desempenho. Com os resultados deste estudo, foi possível quantificar e qualificar a pesquisa relacionada às ferramentas de testes de desempenho existentes na literatura, bem como caracterizar essas ferramentas. Este estudo também propõe uma ferramenta de monitoramento de desempenho chamada PerfMoon, que é versátil e leve, responsável por monitorar determinadas métricas e fornecer essas informações como séries temporais quase em tempo real para outra ferramenta de análise. Este estudo também contempla um benchmark experimental onde a ferramenta desenvolvida tem seus resultados comparados com outra ferramenta.

Palavras-chave: Monitoramento, Software de aplicação, Qualidade de software, Custos, Teste de software, Monitoramento de desempenho, Análise de desempenho, Experimento

List of Figures

Figure 1 – Types of Research Methods	27
Figure 2 – Research Design	28
Figure 3 – Integration of SPE in the software development process	32
Figure 4 – The monitoring process	35
Figure 5 – SMS Process	37
Figure 6 – Search String.	39
Figure 7 – Input approach Venn diagram.	44
Figure 8 – Taxonomy of performance testing tools represented by feature model .	49
Figure 9 – COSMOS - LESSE's Performance Testing Solution.	57
Figure 10 – PerfMoon Architecture	61
Figure 11 – Expected VUser Ramp Up.	68
Figure 12 – CPU Usage - Monitored by PerfMoon	70
Figure 13 – CPU Usage - Monitored by Google StackDriver	71
Figure 14 – Memory Usage - Monitored by PerfMoon	71
Figure 15 – Memory Usage - Monitored by Google StackDriver	71
Figure 16 – Disk I/O Consumption - Monitored by PerfMoon	72
Figure 17 – Disk I/O Consumption - Monitored by StackDriver	73
Figure 18 – Network Throughput Consumption - Monitored by PerfMoon	73
Figure 19 – Network Throughput Consumption - Monitored by StackDriver	74
Figure 20 – Response Time - Monitored by PerfMoon	74
Figure 21 – Response Time - Monitored by StackDriver	75

List of Tables

Table 1 – Research Synthesis.	29
Table 2 – Objective according to the GQM paradigm	38
Table 3 – PICOC structure	38
Table 4 – Search string definition.	39
Table 5 – Tools and references.	43
Table 6 – Tools and workload input approaches.	46
Table 7 – Monitored metrics and Persistence strategies.	47
Table 8 – Related work summary.	55
Table 9 – Test Scenario.	65
Table 10 – SUT Hardware Configuration.	67
Table 11 – Ramp up configuration.	68
Table 12 – Testbed Hardware Configuration.	68
Table 13 – Cosine Application.	75
Table 14 – Schedule.	83

List of acronyms

BPMN Business Process Model and Notation

CNPQ Conselho Nacional de Desenvolvimento Científico e Tecnológico

CR Capture Replay

DdS Distributed Denial of Service Attack

DoS Denial of Service Attack

EC Exclusion Criteria

FR Functional Requirements

IC Inclusion Criteria

LESSE Laboratory of Empirical Studies in Software Engineering

MBT Model-Based Testing

NFR Non-Functional Requirements

QoS Quality of Service

RQ Research Question

SBS Web Service Based Systems

SE Software Engineering

SLA Service Level Agreements

SMS Systematic Mapping Study

SPE Software Performance Engineering

SUT System Under Test

UNIPAMPA Federal University of Pampa

VU Virtual Users

WBS Web Based Systems

Table of Contents

1	INTRODUCTION	23
1.1	Motivation	24
1.2	Objectives	24
1.3	Contribution	25
1.4	Organization	25
2	METHODOLOGY	27
2.1	Introduction	27
2.2	Research Design	28
2.3	Research Synthesis	29
2.4	Chapter Summary	29
3	BACKGROUND	31
3.1	Software Testing	31
3.2	Software Performance Engineering	31
3.3	Performance Testing	32
3.4	Performance Monitoring	33
3.5	Performance Testing Tools	34
3.6	Chapter Summary	35
4	SYSTEMATIC MAPPING STUDY	37
4.1	Protocol	37
4.1.1	Scope and Objective	37
4.1.2	Question Structure	38
4.1.3	Research Question	38
4.1.4	Search Process	38
4.1.5	Inclusion and Exclusion Criteria	39
4.1.6	Quality Assessment Criteria	40
4.1.7	Selection Process	41
4.1.8	Data Extraction Strategy	42
4.1.9	Data Analysis Strategy	42
4.2	Systematic Mapping Study Results	42
4.3	Taxonomy of Performance Testing Tools	48
4.3.1	Input Approaches	48
4.3.1.1	Capture Replay	48
4.3.1.2	Model-Based Testing	50
4.3.1.3	Scripting	50
4.3.2	Load Generator	50
4.3.2.1	Architecture	50

4.3.2.2	Implementation	51
4.3.3	Monitor	51
4.3.3.1	Metrics	51
4.3.3.2	Data Persistence	53
4.3.4	Analysis	53
4.3.4.1	Representation	53
4.3.4.2	Result Analysis	53
4.4	Threats to Validity	54
4.5	Related Work - Systematic Mapping Review	54
4.6	Chapter Summary	55
5	PERFMOON'S DESIGN	57
5.1	Initial Considerations	57
5.2	Aspects of Analysis	57
5.3	PerfMoon Requirements	58
5.3.1	Functional Requirements	58
5.3.2	Non-Functional Requirements	59
5.4	Design Decisions	60
5.5	Architecture	61
5.6	Chapter Summary	62
6	EXPERIMENTAL BENCHMARKING: PERFMOON'S	63
6.1	Initial Considerations	63
6.2	Experimental Benchmarking Design	63
6.3	Collecting Data Evidence	65
6.4	Test Environment	66
6.4.1	System Under Test (SUT) Environment	66
6.4.2	Testbed Environment	67
6.4.3	Environment Toolkit	68
6.5	Experimental Benchmarking Analysis	70
6.5.1	CPU Consumption	70
6.5.2	Memory Consumption	71
6.5.3	Disk I/O Consumption	72
6.5.4	Network Throughput Comparison	72
6.5.5	Response Time	74
6.5.6	Deviation Results	74
6.6	Threats to Validity	75
6.6.1	Internal Validity	76
6.6.2	External Validity	76
6.6.3	Construct Validity	76

6.6.4	Conclusion Validity	77
6.7	Chapter Summary	77
7	CONCLUSIONS AND FUTURE WORK	79
7.1	Conclusions	79
7.2	Lessons Learned	80
7.2.1	Efficient Systematic Mapping Protocol	80
7.2.2	Testing Tools Classification	80
7.2.3	Software Testing	80
7.2.4	Performance Testing	80
7.2.5	Performance Monitoring	81
7.3	Publications	81
7.4	Schedule	82
	BIBLIOGRAPHY	85
	Index	93

1 INTRODUCTION

The process of testing software is an essential activity in Software Engineering. In simplest terms, it amounts to observing the execution of a software system to validate whether it behaves as intended and identify potential malfunctions. Tests are widely used in software projects to ensure quality, by directly scrutinizing the software in execution (BERTOLINO, 2007).

Although there are few works published describing approaches to software performance testing and monitoring, it is nonetheless an extremely significant issue for many projects (WEYUKER; VOKOLOS, 2000). Often, the primary problems that projects report after field release are not system crashes or incorrect system responses, but rather system performance degradation or problems handling required system throughput. When queried, it is not uncommon to learn that, although the software system has gone through extensive functionality testing, it was never really tested to assess its expected performance (WEYUKER; VOKOLOS, 2000).

Myers e Sandler (2004) highlighted, the software testing is a key part for developing quality software products. When compared to other software development life cycle phases, which can represent up to 60% of the total development, software testing activities require many resources, such as time and money.

The purpose of testing includes quality assurance, verification, validation and reliability estimation. To balance team expectations, and due to time-to-market pressures, software projects usually present different points of view in terms of effort investment. For instance, it is common to observe conflicts of interest among project managers, who deal with time schedules, test managers, who deal with quality assurance, and company managers focusing on budgeting. In this sense, empirical works provided many evidences suggesting that time invested in testing saves money in software projects (MYERS; SANDLER, 2004), (PERRY, 2007), (AMMANN; OFFUTT, 2016).

With the high demand for Web systems, such as *e-commerce and corporate applications*, the data flow demand of these systems grows daily, making these systems capable of receiving a large number of requests in a short time.

While this technology is widely expected to enable the interoperation of heterogeneous systems and the reuse of distributed functions, the industry uptake of this technology has been slow (LANGDON, 2003). Some research has revealed that the lack of quality assurance and guarantee is one of the main factors (DEGWEKAR; SU; LAM, 2004).

The behavior of these systems is not only determined by their own software, but also by many other factors, for example, hardware, network and even customer requests. Limited hardware resources or low network bandwidth tend to cause a long response time, many customer requests may be lost or too long.

All these unexpected results are deviations of Web applications behavior from

the requirements. Monitoring certain run-time information about these behavior-related factors is an important issue to ensure the consistency of Web services behavior.

In practice, load tests are rarely an integral part of the development process, see (SHAMS; KRISHNAMURTHY; FAR, 2006). The main reasons besides lack of money or time may be the common prejudice that load tests are very difficult to create and maintain. There are also statistics that report that the benefits of load tests do not outweigh their costs.

1.1 Motivation

For modern Web applications it is critical to remain available and guarantee short response times even if accessed by large numbers of concurrent users. Studies show that response times above one second cause dissatisfaction, which makes users switch to a competitor sooner or later (CHEUNG; LEE, 2005).

Building applications that meet these performance requirements starts with planning and testing these requirements the early stages of the development process. Load tests are carried out to ensure that the required number of customers can be served simultaneously without exceeding a certain response time and performance requirements such as throughput and utilization (BENEDIKT; FREIRE; GODEFROID, 2002).

In computing undergraduate courses, only a few disciplines are allocated for testing activities when compared to other activities in the software development process. Although the test activity has a small participation in the curricular time load, it is commonly found in undergraduate courses in the computing area.

The main motivation of this work was the importance of software testing for the development of high quality and reliable software systems and also the lack of open source tools available to assist in the monitoring process with specific metrics for software available on Web.

1.2 Objectives

The general objective of the proposed work is to specify, design and implement an open source tool that performs monitoring activities on certain Web software metrics, in order to standardize procedures for performing these tasks, independently of the system, minimizing the complexity involved in each one these tasks.

Order to achieve the general objective, the following specific objectives are defined:

- Conduct empirical research through the scientific literature to find and study proposed and applied performance testing tools and methods;
- Specify requirements, architecture, use cases, functionalities, draw up diagrams, design decisions and implement the proposed tool, so that it can add value among

the tools researched and, above all, remedy the weaknesses observed in existing solutions;

- Evaluation the proposed monitoring tool using a experimental benchmark, in comparison with techniques, methods or usual approaches for performance monitoring;
- Evaluate the proposed tool and analyzing the representation power of the characteristics of the monitoring test by expert groups in the field whereby the experimental benchmark, case study and/or research was conducted;
- Documenting and reporting the study results in the scientific publications.

1.3 Contribution

This study is an independent module that will be integrated into a complete performance and monitoring test solution, which is being developed by the Laboratory of Empirical Studies in Software Engineering (LESSE)¹. Additional information about the research group can also be found on the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ)² platform.

The main contribution of this study is the development and implementation of a monitoring tool, called PerfMoon, described in detail in the Chapter 5.

In order to make the main contribution possible, previous studies was conducted, such as the execution of a Systematic Mapping Study SMS. This study mapping of the primary studies in the area of performance and monitoring tests allowed the identification of possible gaps in the area. The conduction of this study also resulted in a taxonomy proposal, used to identify and characterize performance testing tools and monitoring according to their characteristics, described in Section 4.1.

1.4 Organization

This document is organized according to the following:

- **Chapter 2: Methodology:** Details of the the methodology adopted for the execution of this study.
- **Chapter 3: Background:** Details of main concepts related to our work, such as, performance testing and monitoring.
- **Chapter 4: Related Work:** This chapter reviews other work in the area of performance and monitoring testing and reports how they relate to this work.
- **Chapter 5: PerfMoon's Design:** Provides details of the design and implementation of the proposed tool.

¹ LESSE's Website is available at: <<http://lesse.com.br/>>

² CNPQ's Website is available at: <<http://dgp.cnpq.br/dgp/espelhogrupo/5867775896704418>>

- **Chapter 6: Experimental Benchmarking:** This chapter presents the experimental benchmarking performed of this study.
- **Chapter 7: Conclusions:** This chapter presents the publications and general conclusions about this study.

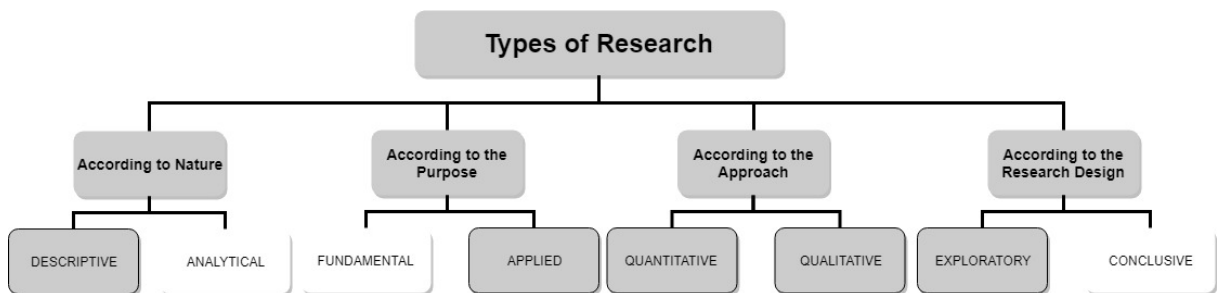
2 METHODOLOGY

This chapter discusses the methodology adopted for the execution of this study. The following sections describe the specific procedures or techniques used to identify, select, process, and analyze information. Section 2.1 lists the sorts of search types. Section 2.2 describes how this research was conducted. Section 2.3 describes how the study was formalized and 2.4 briefly presents chapter information.

2.1 Introduction

According to Wazlawick (2017), studies generally have a chapter for elucidating the methodology. However, analyzing the term semantically, the methodology would be the study of methods. Research methods can be defined as "a systematic scientific procedure of data collection, compilation, analysis, interpretation, and implication pertaining to any problem" (CHAWLA; SODHI, 2011). Types of research methods can be classified into several categories according to the nature, purpose and approach of the study and other attributes.

Figure 1 – Types of Research Methods



Source – Adapted from Kumar (2008)

Figure 1 shows the classification of our research by its nature, purpose, approach and research design. The classification used in this work is represented by the gray classes and are explained in this Section.

According to Kumar (2008), descriptive research usually involves surveys and studies that aim to identify the facts. In other words, descriptive research mainly deals with the "description of the state of affairs as it is at present", and there is no control over variables in descriptive research. This method was used in this work to assist in the description of the characteristics of the monitoring tools already existing in the literature.

The purpose of the study is a applied research. Applied research is also referred to as an action research, problems are analysed from the point of one discipline, generalisations are preferred and reports are compiled in a language of technical language of discipline.

According to the approach two research methods were widely used:

- **Quantitative:** This method was used to describe the auxiliary in the process of collecting numerical data, in the summary of these data and in the collection of inferences from the data;
- **Qualitative:** This method was used to observe and base non-numerical and non-quantifiable information. Any information that can not be analyzed by mathematical techniques has been included in this group.

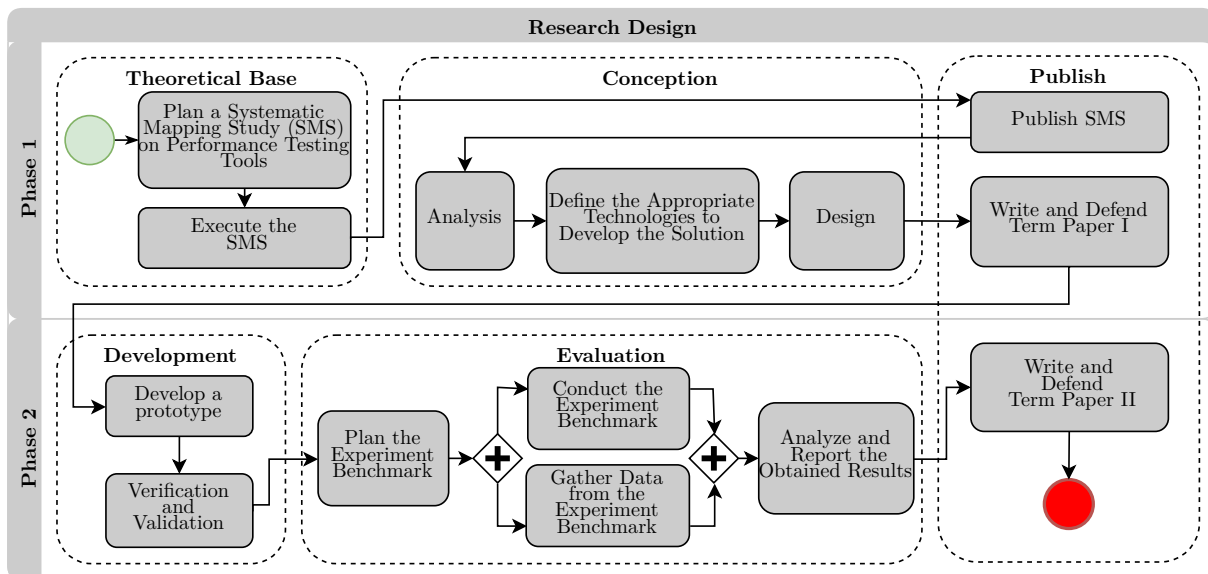
Based on the research design represented in Section 2.2, the type of research method used was exploratory. Exploratory research is conducted to have a better understanding of the existing problem and used to investigate a problem which is not clearly defined.

2.2 Research Design

The structuring the research design is one of the phases of the methodological process and is based on different stages, being the formulation of the problem to be questioned, the conception of hypotheses to be verified, data collection, data tabulation, data analysis, discussion of results, conclusions, writing of the text and presentation of scientific work.

For the conduction of this study, a research design was developed. This design was created from the process management notation known as Business Process Model and Notation (BPMN) and is represented by Figure 2.

Figure 2 – Research Design



Source: author.

The process described in Figure 2 was developed with well-defined activities. The first phase began with the formulation of the theoretical foundation where was need for well-founded definitions. This phase was completed with the end of SMS execution.

In the conception phase, the analysis was performed and the essential requisites were defined and documented, as well as the definition of the appropriate technologies to develop the solution. From those results a solution was designed and can be seen in Chapter 5. Once this is done, the development and implementation of a testable prototype will begin in the development phase, so that the testing can be performed and the prototype evaluated.

The prototype evaluation activities are contained in the evaluation group and include the planning of a controlled experiment, the execution of the experiment parallel to the data collection and, afterwards, a report with the results obtained will be developed and included in term paper II.

2.3 Research Synthesis

Based on the objectives presented in Section 1.2, to formalize this study, the following research synthesis was elaborated:

Table 1 – Research Synthesis.

Subject	Performance Testing
Topic	Performance Monitoring
Research Question	Would the state of practice benefit from a open source performance monitoring tool for Web application to integrate a performance testing solution?
Hypothesis	The performance testing practice would benefit from a open-source performance monitoring tool that is easy to learn, light-weight and easy to integrate.
Main Goal	To develop a performance monitoring tool for Web applications.

2.4 Chapter Summary

This chapter provided an idea of what the methodology is and how this research can be classified. In addition, the established research project has been presented so that it is possible to understand which processes have been executed so far and which ones will still be executed. Chapter 3 provides the theoretical basis necessary for the accomplishment of this work and also assists in understanding the mentioned research process.

3 BACKGROUND

In this chapter, we provide definitions of the terms used in this study in the context of software performance and software performance measurement. Sections 3.1 and 3.2 introduce the main features of software testing and software performance engineering. The Sections 3.4 and 3.5 report the theoretical basis of the work in relation to monitoring and monitoring tools.

3.1 Software Testing

Testing is defined as a process of evaluation that either the specific system meets its originally specified requirements or not (JAMIL et al., 2016). It is mainly a process encompassing validation and verification where the developed system meets the requirements defined by the user. Software testing refers to finding bugs, errors or missing requirements in the developed system or software. Thus, it is an investigation that provides the stakeholders with the exact knowledge about the quality of the product.

Software testing can also be considered as a risk-based activity. An important issue related with the testing process is related to the understanding that testers must have about how to minimize a large number of tests into manageable tests (PATTON, 2005).

Software testing is an important component of software quality assurance. The importance of testing can be considered from life-critical software, *e.g.* flight control, testing which can be highly expensive because of risk regarding schedule delays, cost overruns, or outright cancellation (FELDERER; SCHIEFERDECKER, 2014).

Software testing contains certain levels and steps according to which the person who does the testing differs from level to level (KOREL, 1990). Section 3.2 reports a software-oriented approach that focuses on architecture, design, and implementation choices.

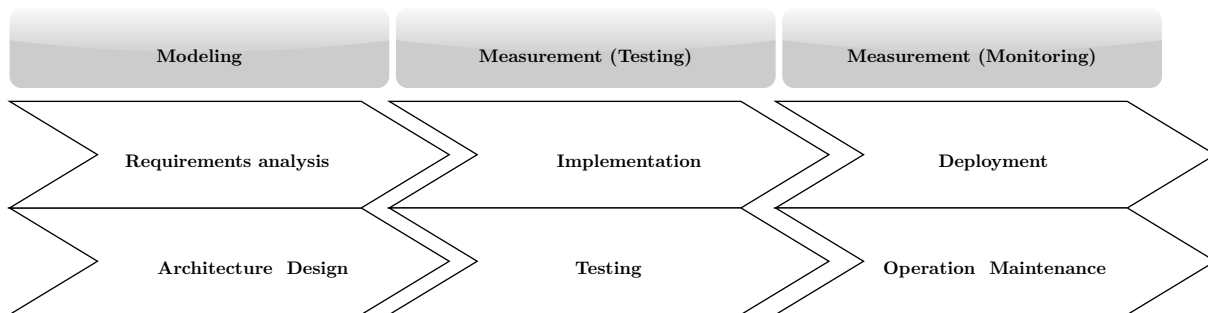
3.2 Software Performance Engineering

The term Software Performance Engineering (SPE) was coined by Smith (1990) for a model-based approach to construct software systems that meet performance requirements. Smith e Williams (2003) applied this approach to object-oriented systems. In their approach, a systematic, quantitative approach to constructing software systems that meet performance objectives is constructed.

Woodside, Franks e Petriu (2007) provides a broader definition of SPE, also including measurement-based approaches. Performance measurement complements and validates performance prediction models, but requires prototypical implementations or earlier versions of the software system.

The SPE process begins early in the software development life cycle and uses quantitative methods to identify satisfactory designs and to eliminate those that are likely to have unacceptable performance before developers invest significant time in implementation. SPE continues through the detailed design, coding and performance and load testing phases to predict and manage the performance of the evolving software as well as monitoring and reporting actual performance versus specifications and predictions and are represented in Figure 3. SPE methods encompass: performance data collection; quantitative performance analysis techniques; prediction strategies; management of uncertainties; data presentation and tracking; performance testing, stress and load testing, model verification and validation; critical success factors; and performance design principles, patterns and anti patterns (FOSTER, 1995).

Figure 3 – Integration of SPE in the software development process



Source: Software Performance Engineering (SPE) approaches adapted from (EHLERS, 2012)

3.3 Performance Testing

Features and functionality supported by a software system is not the only concern. A software application's performance like its response time, reliability, resource usage and scalability do matter. The goal of performance testing is not to find bugs but to eliminate performance bottlenecks. Performance testing is defined as a type of software testing to ensure software applications will perform well under their expected workload (KHAN, 2010).

Performance testing is done to provide stakeholders with information about their application regarding speed, stability, and scalability. More importantly, performance testing uncovers what needs to be improved before the product goes to market. Without performance testing, software is likely to suffer from issues such as: running slow while several users use it simultaneously, inconsistencies across different operating systems and poor usability. To talk about software performance testing it is necessary to understand all the activities involved in evaluating how the system can be expected in the field. This is assessed from the user's perspective and is generally evaluated in terms of throughput, response time, or some combination of the two. Alternatively, the performance test can be used to evaluate the level of system availability (MATHUR, 1991).

Within that structure, there are a number of different goals regarding performance testing and monitoring, in which including:

- The design of test case selection or generation algorithms specifically intended to test for performance criteria rather than functional correctness criteria;
- The definition of metrics to assess the comprehensiveness of a performance test case selection algorithm for a given program;
- The definition of metrics to compare the effectiveness of different performance testing strategies for a given program;
- The definition of relations to compare the relative effectiveness of different performance testing strategies in general; This requires that we are able to say in some concrete way what it means for this performance testing strategy to be better than that one;
- The comparison of different hardware platforms or architectures for a given application.

There are also a number of things that require to be measured when evaluating a software system's performance. Included among these are: resource usage, throughput, stimulus-response time, and queue lengths detailing the average or maximum number of tasks waiting to be serviced by selected resources. Typical resources that need to be considered include: network bandwidth requirements, CPU cycles, disk space, disk access operations, and memory usage (SMITH, 1990).

3.4 Performance Monitoring

Software monitoring involves obtaining the information relating to the state, behavior, and environment of a software system at run time, so as to deal with potential deviations of system behavior from requirements at the earliest possible time. Monitoring is usually carried out in parallel with the system's normal execution, without interrupting its operation. Starting from early 1960s with the advent of debuggers, software monitoring has been widely used for debugging and testing, correctness checking, security and dependability analysis, performance evaluation and enhancement, and system control (SCHROEDER, 1995).

Performance monitoring is the single way that can measure the computer system behaviour in the system's real environment. Monitors collect data that can serve as a basis for computer tuning, computer system model design, workload analysis, model validation, or just finding out more about how the system really works. Performance monitors are either programs that execute on the same system they measure (software monitors)

or independent devices that are attached to the monitored computer through a set of electronic probes (hardware monitors). Each technique has its own advantages and disadvantages. Combining the advantages of software and hardware monitoring techniques has already brought about some increases in both the efficiency and the effectiveness of performance monitoring (WHOLEY; HATRY, 1992).

The necessity of specifying and monitoring the different compositional properties as well as the functional and nonfunctional requirements of Web Based Systems is widely recognized by industry and academia.

Lamanna, Skene e Emmerich (2003) have proposed a Web Service Based Systems monitoring approach with the introduction of the language SLAng. This language is an extension of the existing business process languages. In this language properties are defined as a list of Quality of Service parameters. At the implementation stage QoS parameters are assigned to the target business process, this leads to an intrusive approach. The target servers are required to support these QoS parameters. This approach becomes less extensible and flexible. The approach described in Sahai et al. (2002) creates monitoring agents to monitor the business process. These agents monitor the business process by gathering the network usage information. Another process evaluates the properties for any change in the process. This approach requires the business process to update constantly in order to adopt to new property requirements.

A software monitor can abstract data stored in memory, but it always introduces an additional system overhead. A badly designed software monitor can create an overhead as large as 40% of the total processing time Zheng et al. (2018). A simple hardware monitor can count event occurrences and time event duration and record collected data without the slightest interference with the host system. A more sophisticated monitor can accumulate data in a fashion that enables later output of complete frequency distributions of monitored events, but the domain of events that can be measured by a hardware monitor is limited. Such events have to have a hardware representation that is externally accessible and does not require addressing to make the stored information available. An easy-to-use interface between the system's software and an external hardware monitor would solve many problems (JANES; LENARDUZZI; STAN, 2017).

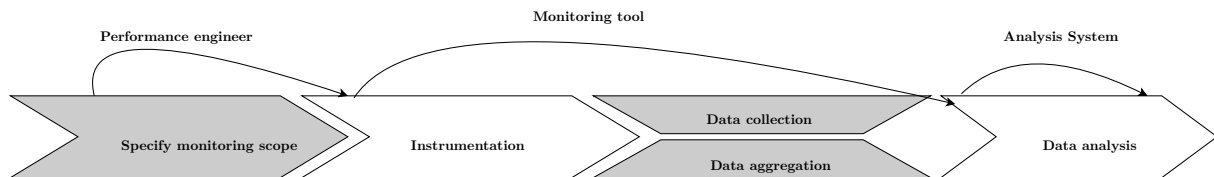
3.5 Performance Testing Tools

Eichelberger et al. (2015) propose a monitoring process common to most monitoring tools (see Figure 4). In this process a performance engineer determines the intended monitoring scope for the SUT. This step is highly dependant on the system and on the intended final analysis. Next, the monitoring tool performs an instrumentation of the SUT and gathers monitoring data. Depending on the monitoring tool, this data can be further aggregated or simply collected for a concurrent or subsequent analysis. This analysis is usually performed by the performance engineer with the help of an analysis system that

is often part of the monitoring tool.

In the case of monitoring tools, the measurement is usually performed during the operation of the live system under real (in contrast to realistic) conditions. Thus, the monitoring results provide a good description of the actual events in the system. But the overhead caused by the monitoring tool has to be minimal, limiting the amount of data retrieved.

Figure 4 – The monitoring process



Source: adapted from (EICHELBERGER et al., 2015)

Performance testing tools address applications and system design problems by testing scalability and reliability. Unlike bug testing tools, performance and load testing tools establish a performance baseline and then attempt to find out performance bottlenecks by adding up stress. Specific tests include spike tests, soak or endurance tests, load tests, and others. The main objective of performance testing applications is to establish benchmarks for a target system. A target might be a server, a Web app, or a group of servers, or a whole network. Performance testing can be monitored in real-time for validity testing. Results should supply root cause analysis and trace bottlenecks. Additionally, performance test tools calculate Service Level Agreements (SLA) compliance and offer an overall view of system resilience (SUBRAYA; SUBRAHMANYA, 2000).

3.6 Chapter Summary

The most important concepts for our work are presented in this chapter. In general, it was necessary to investigate two major domains, namely: (i) Performance Testing and (ii) Performance Monitoring Tools. Among the topics covered, we highlight Section 3.3 and Section 3.5, which presents important definitions for the understanding of this work.

4 SYSTEMATIC MAPPING STUDY

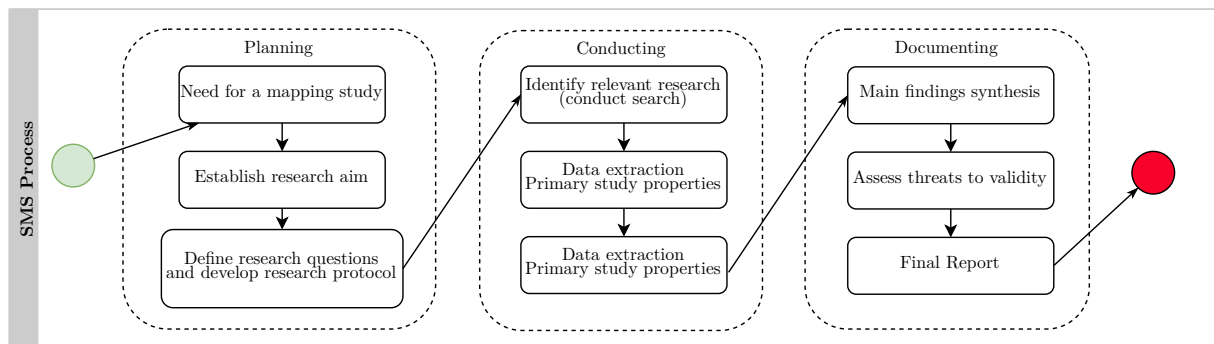
This chapter describes the procedures used to investigate the literature for this study. For this purpose to be achieved was carried out the planning and execution of a Systematic Mapping Study (SMS).

4.1 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 5 shows the steps of the SMS as well as the tasks that were performed in the work conduction.

Figure 5 – SMS Process



Source: Adapted from Engström e Petersen (2015).

4.1.1 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 Koziolk (2008) can be observed in Table 2.

Table 2 – Objective according to the GQM paradigm

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

4.1.2 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 observed in Table 3.

Table 3 – PICOC structure

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

4.1.3 Research Question

The following RQs are defined according to the PICOC question structure established in Section 4.1.2.

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?

4.1.4 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 Library¹, Engineering Village², IEEE Xplore³, ScienceDirect⁴, SCOPUS®⁵ and SpringerLink⁶. To define the search string the terms and synonyms presented in Table 4 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 6.

Table 4 – 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 6 – 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))
```

4.1.5 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>>

4.1.6 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.

4.1.7 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 4.1.5);
- (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 4.1.5);
- (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 4.1.6), 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.

4.1.8 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 extraction⁷

The data presented here were manipulated using the Thoth⁸ 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.

4.1.9 Data Analysis Strategy

The data was tabulated to show: The list of published tools, its licensing and their source in Table 5 (addressing **RQ1.**); The list of published tools, supported input approaches of each and its categorization in Table 6 (addressing **RQ2.1.**); The list of published tools, their quality attributes including monitored metrics and its categorization in Table 7 (addressing **RQ2.2.**); The list of published tools, and the persistence strategies of each tool in Table 7 (addressing **RQ2.3.**).

4.2 Systematic Mapping Study Results

In this section we discuss the answers to our RQs (see Section 4.1.3). 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 5 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 them 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 5 – 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 (LTTTC)	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

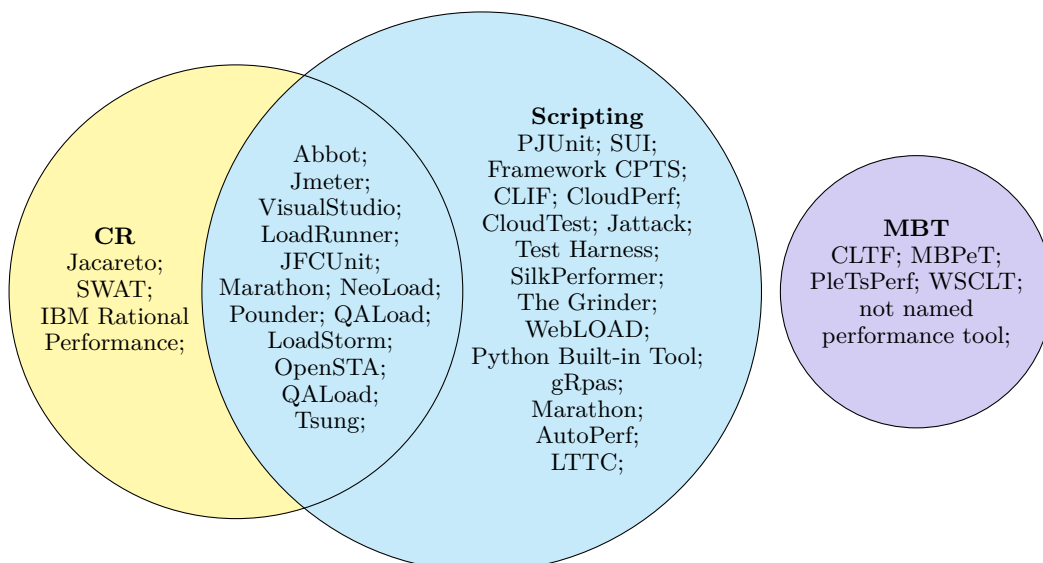
adopting distinct features and strategies, it is perceived that tools developed for this purpose make use of an already consolidated architecture.

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 8 the groups and elements of our taxonomy are presented and explained collectively in Section 4.3, 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 7 and Table 6 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 7 – 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 4.3.3.1. 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 7, 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 7). Finally, as for database management systems, no patterns or preferences were identified.

Table 6 – Tools and workload input approaches.

Tool Name	CR Capture Replay	MBT					Scripting Language																
		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:** Sequential Action Model; **SWM:** Stochastic Workload Model; **UML AD:** UML Activity Diagram; **UML UC:** UML Use Case diagram.

Table 7 – Monitored metrics and Persistence strategies.

Tool Name	Monitored Metrics						Persistence strategy												
	Application						SUT Resources												
	Response time	Hits per sec.	Responses per second	Transactions per second	Transaction success rate	#Virtual Users	Total test time	CPU	Memory	I/O	Network	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.

4.3 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.

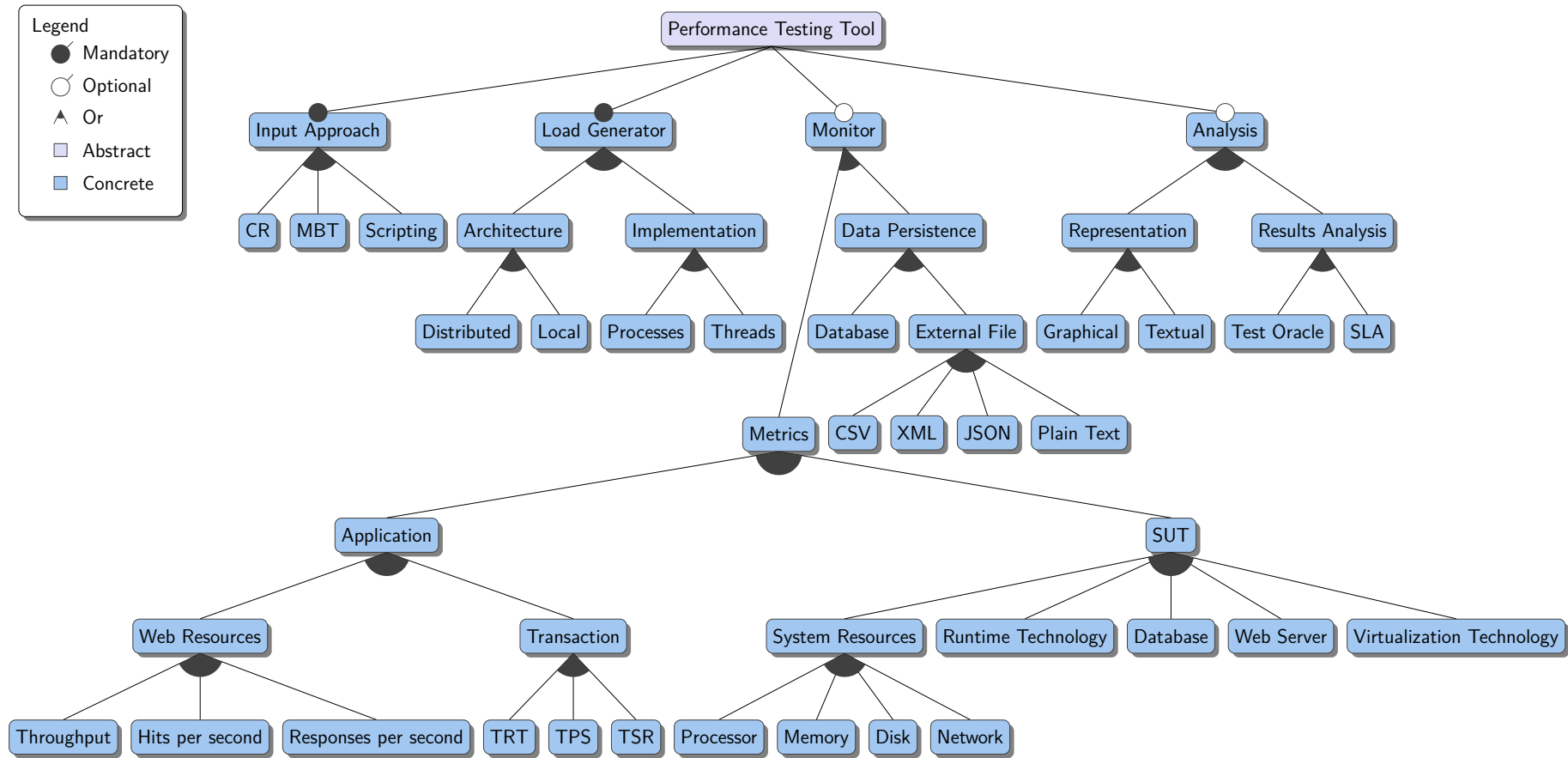
4.3.1 Input Approaches

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

4.3.1.1 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 8 – Taxonomy of performance testing tools represented by feature model



Source: the author.

4.3.1.2 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.

4.3.1.3 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).

4.3.2 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.

4.3.2.1 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, then he has to go through each slave node to make the modifications.

4.3.2.2 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.

4.3.3 Monitor

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

4.3.3.1 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.

4.3.3.2 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.

4.3.4 Analysis

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

4.3.4.1 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.

4.3.4.2 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).

4.4 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.

4.5 Related Work - Systematic Mapping Review

This section summarizes main contributions from related works shown in Table 8.

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 8 – 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 overview of tools and tools taxonomy	not specified	2000 - 2014	not specified	1993 - 2013
Contribution	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

4.6 Chapter Summary

This chapter presented an SMS on performance testing tools. The developed protocol and its execution was described in Section 4.1, its results, the tools found, and the answers to the research questions are discussed in Section 4.2. The main contribution of this mapping is described in Section 4.3 in the form of a taxonomy of performance testing tools.

5 PERFMOON'S DESIGN

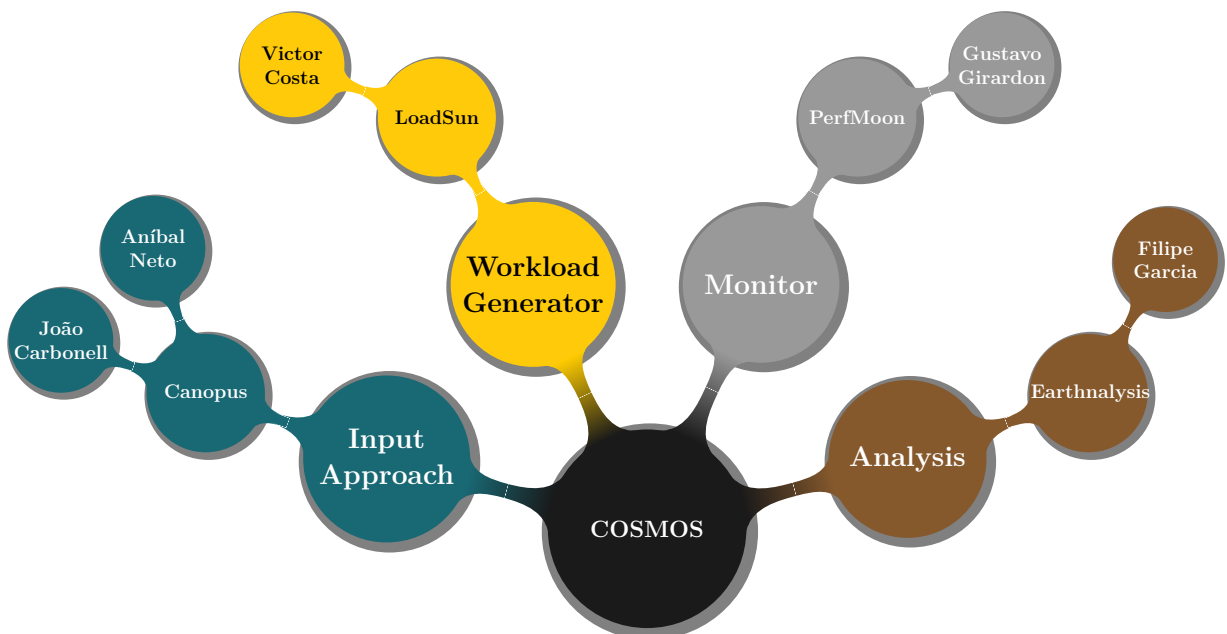
This chapter describes the development of the solution and the creation process behind it.

5.1 Initial Considerations

The PerfMoon tool is designed for the purpose of being an independent module and will be available from the official repository¹. This module will compose a larger and more robust solution in the scenario of performance and monitoring tests. This complete solution is the product of existing projects in the research group LESSE of Federal University of Pampa (UNIPAMPA).

The integration of these modules will enable the modeling of complete test cases in a textual or graphic way, through Canopus DSL proposed by (BERNARDINO; ZORZO; RODRIGUES, 2016). The artifact generated by this DSL will compose the data input required to generate load through the LoadSun tool. The tool developed in this work, PerfMoon, will be responsible for monitoring certain metrics and for generating data that can be analyzed by the Earthanalysis tool. The complete solution called COSMOS can be seen in Figure 9, where the names of the authors are also represented.

Figure 9 – COSMOS - LESSE's Performance Testing Solution.



Source: the author.

5.2 Aspects of Analysis

When a system is in operation, various activities and events happen to the system and its operating environment, while the system states also undergo frequent changes.

¹ PerfMoon's Repository is available at: <<https://github.com/ProjetoDSL/PerfMoon>>

For all events and states relevant to a service to be monitored, we need to know first which of them will affect the service behavior.

Although the monitoring tools have different behaviors and data structures, as can be seen in Table 7, some common characteristics were investigated and mapped among these tools that made possible the design of a tool compatible with these several solutions. In addition to investigating existing tools, framework proposals were also evaluated, as can be seen in Table 6.

From this study was possible to perform an analysis and from this analysis, it was possible to design and implement the tool solution called PerfMoon, which includes Web software monitoring features. The artifacts were generated, considering the functional, non-functional requirements and the definition of these requirements defined the use cases to be designed and implemented in the tool.

5.3 PerfMoon Requirements

The functional and non-functional requirements were categorized according to the functionalities predicted for the tool and described in two groups according to their approach, the first approach refers to metrics directly related to the application, while the second presents metrics related to the features of the system under test (SUT).

5.3.1 Functional Requirements

- **FR01: *Throughput*:** The tool must monitor the amount of server throughput (received and sent bytes) during each second of the load test scenario run. The system throughput or aggregate throughput is the sum of the data rates that are delivered to all terminals in a network. Throughput is essentially synonymous to digital bandwidth consumption; it can be analyzed mathematically by applying the queuing theory, where the load in packets per time unit is denoted as the arrival rate and the throughput, where the drop in packets per time unit, is denoted as the departure rate.
- **FR02: *Hits Per Second*:** The tool must monitor the number of hits per second the server. Hits per second refers to the number of HTTP(s) requests sent by the user(s) to the Web server in a second.
- **FR03: *Processor (CPU)*:** The tool must monitor the percentage of processor (CPU) utilization of the server. This measurement will occur with the following metrics:
 - ***Total CPU*:** This metric is responsible for reporting the result of the sum of the system cpu and user cpu usage metrics.

- **System CPU:** Shows the amount of CPU time used by the kernel. The kernel is responsible for low-level tasks, like interacting with the hardware, memory allocation, communicating between OS processes, running device drivers and managing the file system. Even the CPU scheduler, which determines which process gets access to the CPU, is run by the kernel.
 - **User CPU:** One level up, the “user” CPU state shows CPU time used by user space processes. These are higher-level processes, like your application, or the database server running on your machine. In short, every CPU time used by anything else than the kernel is marked “user”, even if it wasn’t started from any user account.
 - **Idle CPU:** The “idle” CPU state shows the CPU time that’s not actively being used. Internally, idle time is usually calculated by a task with the lowest possible priority.
- **FR04: Memory:** A tool must monitor the percentage of server memory utilization. This measure occurs with the following measures:
 - **Total Memory:** The amount of memory that can be used by programs. The value is obtained by subtracting a few reserved bits and the kernel binary code from the amount of physical RAM available on the system.
 - **Available Memory:** The estimated amount of memory that can be used for starting new programs without swapping.
 - **Used Memory:** A value calculated by subtracting the amount of Buffers, Cache, and Free memory from the Total memory.
 - **FR05: Disk I/O:** The tool should allow writing of the disk input and output logs. Disk I/O includes read or write or input/output operations (defined in KB/s) involving a physical disk. In simple words, it is the speed with which the data transfer takes place between the hard disk drive and RAM, or basically it measures active disk I/O time.
 - **FR06: External Notification:** The solution must allow the load generator tool (LoadSun’s) to communicate and report the exact start and end time of the load test. This will allow the tool to start or end load test log recording.
 - **FR06: Log Format:** The tool should allow the user to choose which format to save their logs. Initially the tool should work with json and xml format.

5.3.2 Non-Functional Requirements

- **NFR01:** The tool must efficiently organize the way it will compress the log files.

- **NFR02:** The solution must be possible to be integrated with other modules of COSMOS Performance Testing Solution developed by the LESSE group.
- **NFR03:** The solution should use a minimal system resources as possible under test.
- **NFR04:** The solution must be possible to be integrated with other modules. In order to fully evaluate the performance of a system the solution must take into account the implementation and the communication protocols utilized by the other modules developed at the LESSE group.

5.4 Design Decisions

This section describes the decisions taken to carry out this study and the development of this tool.

DD01: Programming Language: Python is a highly functional programming language it can do almost what other languages can do with comparable speed. It is used to make data analysis, create GUIs and websites. This language has an excellent interactive shell and has a large collection of open source packages, a very simple syntax and it takes much less time to write and debug by being simple and readable. These are the main reasons for choosing the language.

DD02: Architecture: The architecture was based on a server-client model widely used in the most diverse current systems. One of the main advantages of this decision is proper management, which makes file management a simple task because the files are all stored in the same location. The designed architecture is best described in Section 5.5 and a simplified version can be seen in Figure 10.

DD03: License: Open source solutions often have thriving communities around them, bound by a common drive to support and improve a solution that the community benefit from. The global communities united around improving these solutions introduce new concepts and capabilities faster, better, and more effectively than internal teams working on proprietary solutions.

DD04: Data Persistence: This setting are related to part of the monitored information that the user wishes to save to the database. With the high volume of data that can be monitored and generated, it is of the utmost importance that such data be organized and made available to other tools.

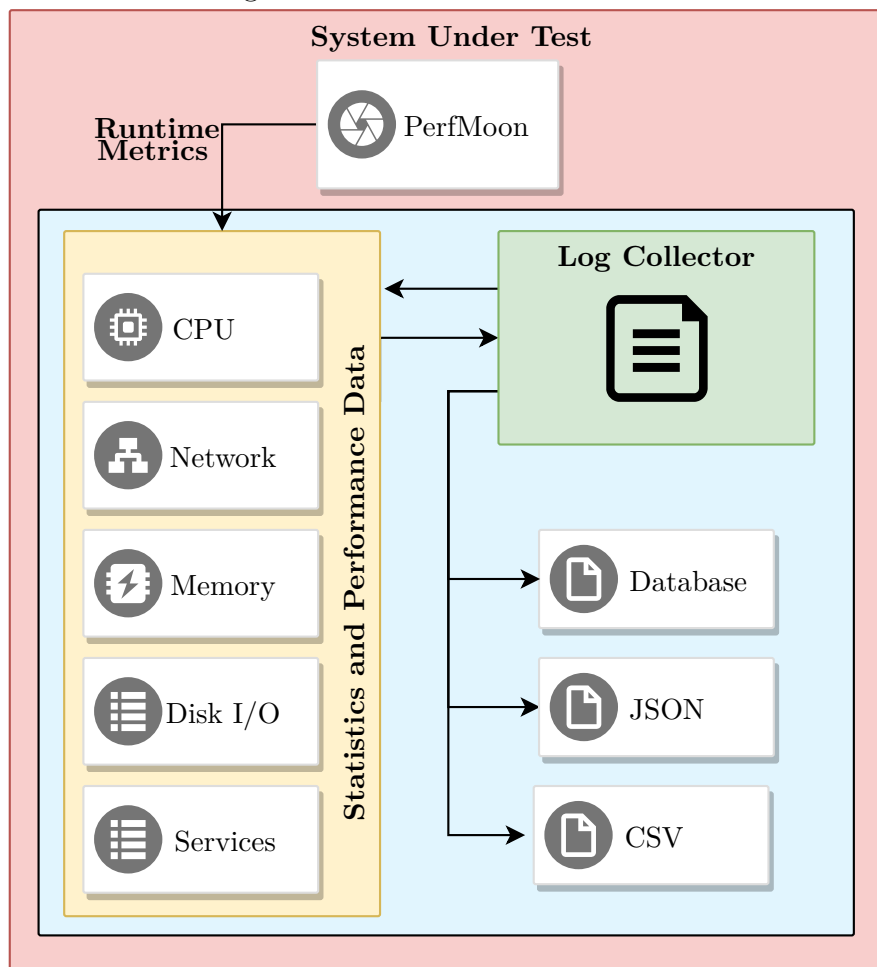
DD05: Data Integrity: A fundamental component of information security. In its broadest use, “data integrity” refers to the accuracy and consistency of data stored in a database. All data characteristics must be correct - including monitoring rules, metrics, and definitions - for the data to be complete.

5.5 Architecture

The architecture of the PerfMoon tool is designed to work in a simplified way. The monitoring tool is designed to provide as much information as possible in a short time through a web-based interface. The tool can dynamically adapt how information is displayed or how it will be saved.

Currently the tool uses the psutil library to get exact system information and the log writing settings are dynamic allowing the user to easily change their metrics. As can be seen in Figure 10, some components of the simply designed architecture are presented.

Figure 10 – PerfMoon Architecture



Source: author.

Measurements can be initiated either by the monitoring server (depending on system configuration) or it can be initiated by an external program (or script) residing on the monitored resource.

When a monitoring agent is initiated for collecting metrics, it measures the relevant metric values from the monitored components and passes them on to the monitoring server. Depending on the configuration of the system, the server may send alerts to interested parties on occurrence of an event.

5.6 Chapter Summary

This chapter reported on how the decisions applied to this project were taken. In the Section 5.1 the main purpose of the tool is discussed together with motivation. In Section 5.2 the design decisions are reported. Section 5.3 is responsible for describing the functional and non-functional requirements of the tool. Finally, Section 5.4 and Section 5.5 report the decisions made for the project execution as well as the proposed architecture for PerfMoon.

6 EXPERIMENTAL BENCHMARKING: PERFMOON’S

This chapter is organized as follows: A Section 6.1 describes the initial considerations of this experiment. Section 6.2 describes how the experiment was designed, executed and how the data will be represented. In Section 6.3 we can see what was monitored during the experiment. Section 6.4 describes the test environment and its settings. Finally the final results are presented in Section 6.5.

6.1 Initial Considerations

In this experimental benchmark we followed the model proposed by Robert K. Yin (YIN; SAGE., 2003). The main objective of this experiment is to compare results obtained by monitoring a server with a Web application. This server was monitored using two distinct tools. The first tool used was StackDriver¹. The StackDriver tool is a business tool owned by Google Company. StackDriver aggregates infrastructure metrics, logs, and events, providing developers and operators with an advanced set of observable signals that streamline root cause analysis and reduce average resolution time. StackDriver does not require extensive integration or multiple dashboards nor does it require developers to use a specific cloud provider. The second tool used was PerfMoon. This tool was developed throughout this work and has its development details described in Section 5.3.

These monitoring tools were employed to complement the reporting capabilities of performance test tools. In addition, these monitoring tools can be used to monitor system performance on an ongoing basis and to alert system administrators to lowered levels of performance and higher levels of system errors and alerts. These tools may also be used to detect and notify in the event of suspicious behavior (such as Denial of Service Attack (DoS) and Distributed Denial of Service Attack (DdS)), scan server logs and compile metrics from them.

6.2 Experimental Benchmarking Design

Load testing, in general, refers to the practice of assessing the system behavior under load (BEIZER, 1984). Load refers to the rate of the incoming requests to the system. A load test usually lasts for several hours or even a few days. Load testing requires one or more load generators which mimic clients sending thousands or millions of concurrent requests to the application under test. During the course of a load test, the application is monitored and performance data along with execution logs are stored. Performance data record resource usage information such as CPU utilization, memory, disk I/O and network traffic. Execution logs record the run time behavior of the application under test.

¹ Google StackDriver is available at: <<https://app.google.stackdriver.com/>>

To obtain monitoring data, this experiment benchmark planned to perform two load tests on the monitored system. The first test will be monitored by the PerfMoon's. The second test will be monitored by the StackDriver tool.

These load tests will be generated by the LoadSun tool (COSTA, 2019) that was designed to work as a standalone module in a more complete solution called COSMOS performance testing solution and can be seen in Figure 9. With the educational aspects and the idea of integration in mind the workload generation solution was designed to work in a simple way and with a low entry barrier, however, seeking to maintain the important characteristics of a workload generator, generating workloads at a low cost of computational resources and be adaptable to different types of workload models to support the different types of performance tests.

The result data of these tests will be stored in log files. These log files will be processed and analyzed to demonstrate the dynamic behavior of the monitored system. Results will also be compared against each other to prove that PerfMoon successfully performs monitoring. This will allow the experiment to prove that the PerfMoon tool is functional. When comparing the results of this tool with the proprietary Google Stack-Driver tool, we can say that the monitoring results are consistent.

The results will also be analyzed mathematically with the rating of each scenario tested by the degree of performance deviation between the previous and current runs.

Cosine similarity metric finds the normalized dot product of the two attributes. By determining the cosine similarity, we would effectively try to find the cosine of the angle between the two objects. The cosine of 0 is 1, and it is less than 1 for any other angle.

Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in 0 and 1. One of the reasons for the popularity of cosine similarity is that it is very efficient to evaluate, especially for sparse vectors.

If the two distributions are very similar, the cosine distance is close to 1. If they are very different, the cosine distance is close to 0. As deviation is the opposite of similarity, we use the following formula to calculate the deviation:

$$\cos(t, e) = \frac{te}{\|t\|\|e\|} = \frac{\sum_{i=1}^n t_i e_i}{\sqrt{\sum_{i=1}^n (t_i)^2} \sqrt{\sum_{i=1}^n (e_i)^2}}$$

Note that $t(x)$ and $e(x)$ correspond to the number of instances in the previous and current runs which have response time x for a particular scenario.

The technical specifications of the Web application used and the server can be viewed in Section 6.4.1. The benchmark traffic was defined by common scenarios most commonly encountered in real-life deployments. These scenarios represent virtual users who can access their accounts, search for products, add them to their cart and finally make the purchase. The test scenario details that will be run and monitored can be

Table 9 – Test Scenario.

Script	Action Name	Think time (s)	Request Method	API Route
1	Login	10	POST	/public/login
	Search Request	5	GET	/public/products/search
	Product Detail	15	GET	/public/product/
	Order Product	5	POST	/public/product/order
	Cart	5	GET	/public/cart
	Check out	15	POST	/public/checkout
2	Home Page	5	GET	/public/home
	New Products	10	GET	/public/products/new
	Best Sellers	10	GET	/public/products/bestsellers
	Product Detail	15	GET	/public/product/
	Search Request	10	GET	/public/products/search

viewed in Table 9.

6.3 Collecting Data Evidence

A typical load test uses one or more load generators that simultaneously send requests to the System Under Test (SUT). During this type of test the system behavior data like execution logs and various metrics are collected. Execution logs record software activities and errors. Execution logs are generated by debug statements that developers insert into the source code to record the runtime behavior of the SUT. Metrics can be web application related (*e.g.*, number of passed/failed requests) or sut related (*e.g.*, resource usage information like CPU utilization, memory, disk I/O and network traffic or the end-to-end response time).

Two artifacts are recorded during a load test: execution logs and performance metrics. Execution logs record from application related metrics (*e.g.*, "User authentication successful) and errors (*e.g.*, "Fail to retrieve product"). Performance metrics record the system's resource usage like CPU, memory, and disk I/O. Performance metrics be collected by resource monitoring tools with very little overhead. The information from execution logs and performance metrics complement each other, as over the course of a load test, execution logs record the system behavior and performance metrics keep track of the system resource utilization.

To conduct this experiment benchmark and to be able to compare the monitoring results of the two tools, only a few monitoring metrics were selected to be monitored. The selected metrics have been grouped and are detailed below:

- **Host-Based Metrics:** Any metric involved in assessing the health or performance of an individual machine, currently disregarding your application's batteries and services. Monitored metrics:

- **CPU Usage;**

- *Memory Usage*;
 - *Disk I/O Usage*.
- ***Application Metrics***: These are metrics concerned with units of processing or work that depend on the host-level resources, like services or applications. The specific types of metrics to look at depends on what the service is providing, what dependencies it has, and what other components it interacts with. Metrics at this level are indicators of the health, performance, or load of a Web application. Monitored metrics:
 - *Responses Time*;
 - *Transactions Time*;
 - *Hits Per Second*.
 - ***Network and Connectivity Metrics***: Gauges of outward-facing availability, but are also essential in ensuring that services are accessible to other machines for any systems that span more than one machine. Networks should be checked for their overall functional correctness and their ability to deliver necessary performance. Monitored metrics:
 - *Bandwidth Utilization*;
 - *Server Latency*.

6.4 Test Environment

This section describes technical details of the test environment. Hardware configurations are presented as well as configurations in third party software. The test environment has been configured as required by the tested Web application.

6.4.1 SUT Environment

Public servers are owned and operated by third parties; they deliver superior economies of scale to customers, as the infrastructure costs are spread among a mix of users, giving each individual client an attractive low-cost, “Pay-as-you-go” model. All customers share the same infrastructure pool with limited configuration, security protections, and availability variances. These are managed and supported by the cloud provider. One of the advantages of a public server is that they may be larger than an enterprises cloud, thus providing the ability to scale seamlessly, on demand.

The Google service called Compute Engine delivers (acronym-next-pages’s) running in Google’s data centers and worldwide fiber network. Compute Engine’s tooling and workflow support enable scaling from single instances to global, load-balanced cloud

Table 10 – SUT Hardware Configuration.

CPU	product width	Intel(R) Xeon(R) CPU 2 Core @ 2.30GHz 64 bits
Memory	size	3840MiB
PCI	product width clock	Intel 440FX - 82441FX PMC [Natoma] 32 bits 33MHz
Disk	product size	Google PersistentDisk 10GiB (10GB)
Connection	provider download upload	Google Cloud 1274.13 Mbit/s 987.37 Mbit/s
OS	version	Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1047-gcp x86_64)

computing. Compute Engine’s acronym-next-pages’s boot quickly, come with persistent disk storage, and deliver consistent performance. The virtual servers are available in many configurations, including predefined sizes or the option to create custom machine types optimized for specific needs. Flexible pricing and automatic sustained-use discounts make Compute Engine the leader in price/performance.

To perform this experiment a virtual machine was instantiated in the Compute Engine module of Google Company. Hardware configurations can be observed on the Table 10.

6.4.2 Testbed Environment

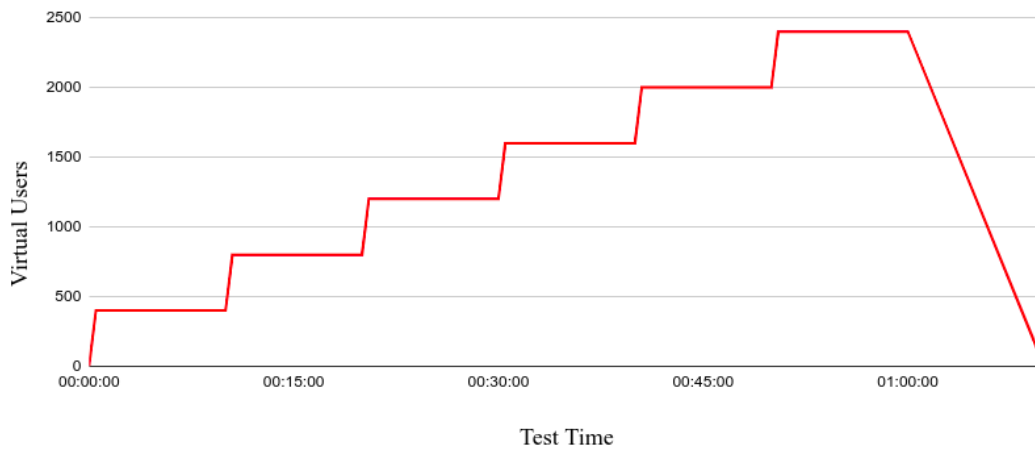
The hardware used to create the load test consists of a robust server providing enough processing power to produce the requested loads. This hardware has been temporarily allocated only for the execution of these load tests. For the execution of this experiment benchmark it was necessary that the monitored servers receive certain workloads. As mentioned in Section 6.1, the LoadSun tool was responsible for the workload used in the test. The test will run for 1 hour and 10 minutes, in this period it will start with 400 VUs and maintain this load for 10 minutes, then it will add another 400 VUs and so on, until 2400 VUs have been kept for 10 minutes then it will start ramping down for another 10 minutes until it reaches 0 VUs and the test ends. Table 11 presents the information related to load generation while Figure 11 shows a model of how the test was designed.

The server configuration used for load generation can be viewed in Table 12, while the test scenario is described in Table 9.

Table 11 – Ramp up configuration.

Total VUs	Test Time	
0	0 min.	Test Start
		Ramp Up
400	10 min.	Ramp Up
800	20 min.	Ramp Up
1200	30 min.	Ramp Up
1600	40 min.	Ramp Up
2000	50 min.	Ramp Up
2400	60 min.	Ramp Down
0	70 min.	Test End

Figure 11 – Expected VUser Ramp Up.



Source: The author

Table 12 – Testbed Hardware Configuration.

CPU	product	Intel(R) Xeon(R) CPU 1 Core @ 2.30GHz
	width	64 bits
Memory	size	3840MiB
PCI	product	Intel 440FX - 82441FX PMC [Natoma]
	width	32 bits
	clock	33MHz
Disk	product	Google Persistent Disk
	size	10GiB (10GB)
Connection	provider	Google Cloud
	download	661.96 Mbit/s
	upload	332.00 Mbit/s
OS	version	Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1047-gcp x86_64)

6.4.3 Environment Toolkit

Environments Toolkit consist of a collection of tools and are intended primarily to support the coding and testing phase of the software development cycle. They provide

little environment-defined control or management over the ways in which the tools are applied. The toolkit approach starts with the operating system and adds coding tools such as a compiler, editor, assembler, linker, and debugger, as well as tools to support large-scale software development tasks such as version control and configuration management.

To carry out the monitoring tests it was necessary to perform certain software installations. Below are all the third party software needed to perform the experiment benchmark.

- ***Linux² - The operating system:*** Ubuntu is a complete Linux operating system, freely available with both community and professional support. Ubuntu community is built on the ideas enshrined in the Ubuntu Manifesto: that software should be available free of charge, that software tools should be usable by people in their local language and despite any disabilities, and that people should have the freedom to customize and alter their software in whatever way they see fit (NEIL, 2016).
 - ***Version used in the experiment:*** Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1047-gcp x86_64).
- ***Apache³ - The web server:*** Apache is the most widely used web server software. Developed and maintained by Apache Software Foundation, Apache is an open source software available for free. It runs on 67% of all web servers in the world. It is fast, reliable, and secure (LIU; IFTIKHAR; XIE, 2014). It can be highly customized to meet the needs of many different environments by using extensions and modules.
 - ***Several Apache modules were used, here are listed the main modules:*** core_module, watchdog_module, http_module, log_config_module, logio_module, fastcgi_module.
 - ***Apache Version:*** Apache/2.4.18 (Ubuntu) mod_fastcgi/mod_fastcgi-SNAP-0910052141
- ***PHP - The programming language:*** PHP⁴: Hypertext Preprocessor (or simply PHP) is a general-purpose programming language originally designed for Web development. PHP code may be executed with a command line interface (CLI), embedded into HTML code, or used in combination with various web template systems, Web content management systems, and web frameworks. PHP code is usually processed by a PHP interpreter implemented as a module in a webserver or as a Common Gateway Interface (CGI) executable (VASWANI, 2009). The Web server

² Ubuntu is available at: <<http://releases.ubuntu.com/16.04/>>

³ Apache is available at: <<https://www.apache.org/>>

⁴ PHP is available at: <<https://www.php.net/>>

outputs the results of the interpreted and executed PHP code, which may be any type of data, such as generated HTML code or binary image data.

- **Version used in the experiment:** PHP 7.2.24-1+ubuntu16.04.1+deb.sury.org+1 (cli) (built: Oct 24 2019 18:28:51) (NTS).

6.5 Experimental Benchmarking Analysis

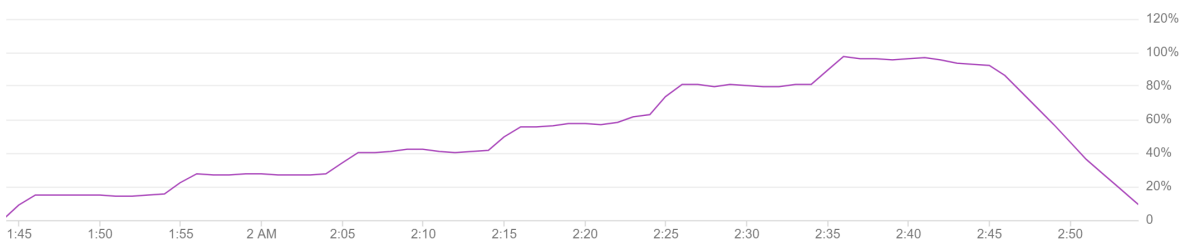
The results of this experiment benchmark are presented in this section. The monitoring details of the tools used (PerfMoon and StackDriver) are compared to each other and detailed in the following sections. Two load tests were performed, the former was monitored with the experimental treatment - PerfMoon -and the later with the control treatment - Google StackDriver tool.

6.5.1 CPU Consumption

CPU utilization refers to a computer's usage of processing resources, or the amount of work handled by a CPU. Actual CPU utilization varies depending on the amount and type of managed computing tasks. Certain tasks require heavy CPU time, while others require less because of non-CPU resource requirements.

During the monitoring of the first test, using the PerfMoon tool, it was noticed that the CPU utilization had a linear increase reaching the peak utilization with approximately 50 minutes of testing. At this time of testing the load of virtual users accessing the system was the maximum set by the script as seen in Table 11. Maximum CPU utilization reached 97.33% and the overall average utilization during the test was 52%.

Figure 12 – CPU Usage - Monitored by PerfMoon



In the second test monitoring using the StackDriver tool, the maximum CPU utilization was 89% and the overall average CPU utilization during the test was 48%. The CPU utilization data during the tests can be viewed in Figure 12 and Figure 13 respectively.

Figure 13 – CPU Usage - Monitored by Google StackDriver



6.5.2 Memory Consumption

During the load tests performed under the system it was possible to notice that the running processes do not need to perform a large memory allocation volume. When constant executions of dynamic processes occur and these are not always cached, processing occurs on the CPU Section 6.5.1. Memory is not the best way for execute dynamic processes in Web applications.

During monitoring with the PerfMoon tool the maximum memory utilization was 23.2% and the overall average was 9.5%.

Figure 14 – Memory Usage - Monitored by PerfMoon



In the monitored test with the StackDriver tool, reported memory utilization was double that reported by the PerfMoon tool. In this monitoring the maximum memory utilization reached 9.8% while the total test average was 5.1%.

Figure 15 – Memory Usage - Monitored by Google StackDriver

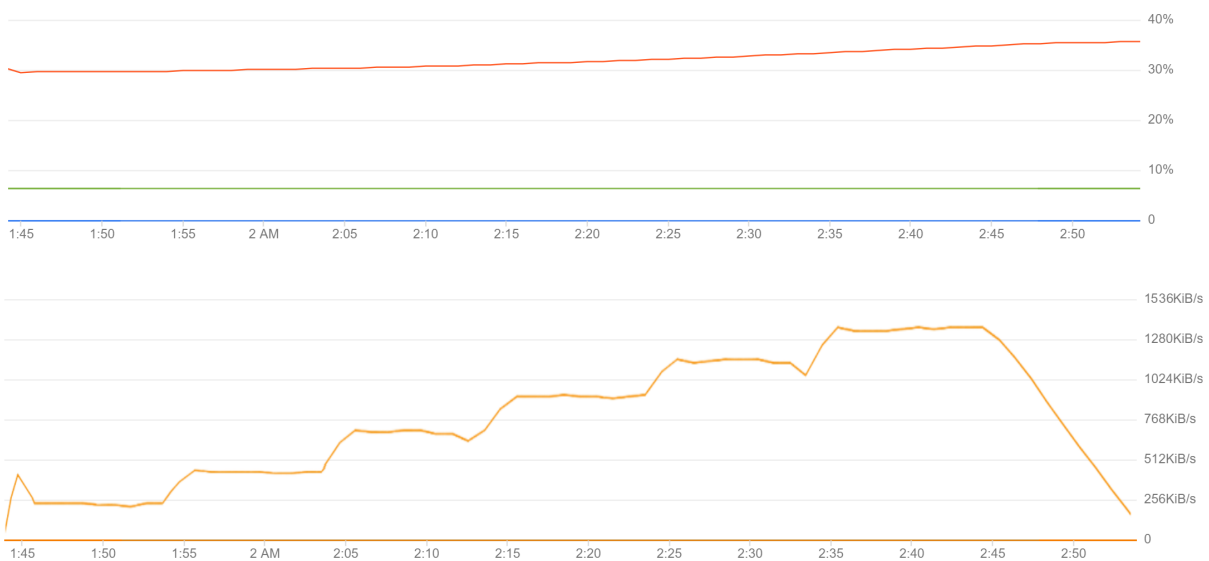


6.5.3 Disk I/O Consumption

As non-volatile storage media hard disk drives and solid state drives are both particularly safety-relevant and performance-critical components within the server environment. Since such an individual storage medium has in comparison with server components, such as the processor or main memory, a very high access time, particular importance is attached to the sizing and configuration of disk subsystems (AHMAD, 2007). On account of the plethora of different application scenarios there is a very large number of configuration options in particular for disk subsystems. It is therefore also not possible to assess all the aspects of a disk subsystem with a single performance counter.

In the monitoring process it was possible to notice that both tools have a similar performance. In monitoring using the PerfMoon tool the maximum disk usage rate was 35.5% and the overall average was 32.8%. During this testing and monitoring the disk write speed reached about 1380KiB/s. This metric is directly linked to the speed with which the PerfMoon tool audits your log files. Figure 16 reports this data graphically.

Figure 16 – Disk I/O Consumption - Monitored by PerfMoon

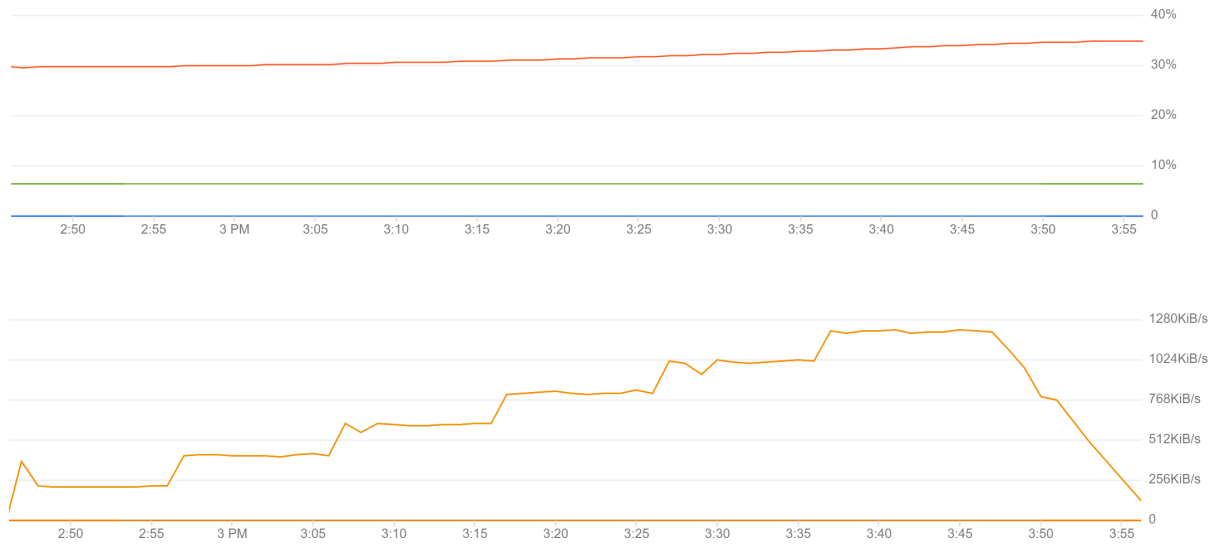


With the StackDriver tool, the maximum disk usage rate was 34.6% with an overall average of 31.6%. The write speed peaked at 1195KiB/s as shown in the following Figure 17.

6.5.4 Network Throughput Comparison

Network throughput is usually represented as an average and measured per second, or in some cases as data packets per second. Throughput is an important indicator of the performance and quality of a network connection. A high ratio of unsuccessful message

Figure 17 – Disk I/O Consumption - Monitored by StackDriver



delivery will ultimately lead to lower throughput and degraded performance. In the monitoring descriptions we will use the acronyms TX and RX which are abbreviations for Transmit and Receive, respectively.

In the tests performed, the received rate captured by the PerfMoon tool peaked at 274.9KiB/s and an overall average of 160.2KiB/s during the test, while the transmit rate peaked at 268KiB/s and the overall average of 148KiB/s. With the StackDriver tool the maximum received rate monitored by the tool was 238.8KiB/s and the overall average was 140KiB/s. The transmit rate peaked at 223.6KiB/s and an overall average of 127KiB/s. Figure 18 and Figure 19 report these results. Network bandwidth consumption was also monitored. During the 1 hour and 10 minute tests, the PerfMoon tool captured 801.1 Megabytes of data sent and received, while the StackDriver tool reached 760.5 Megabytes.

Figure 18 – Network Throughput Consumption - Monitored by PerfMoon

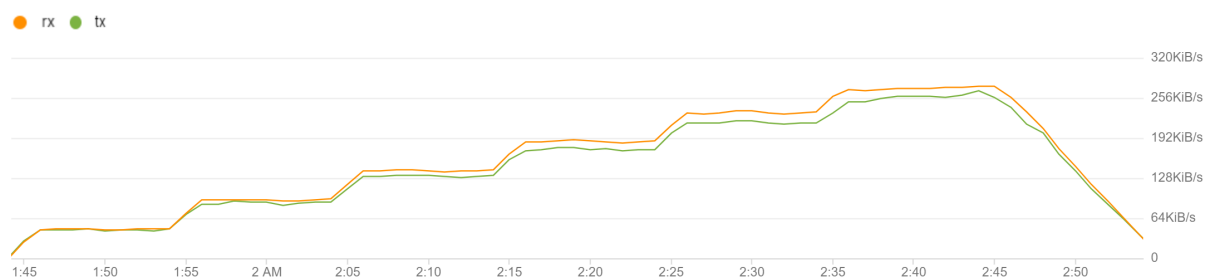
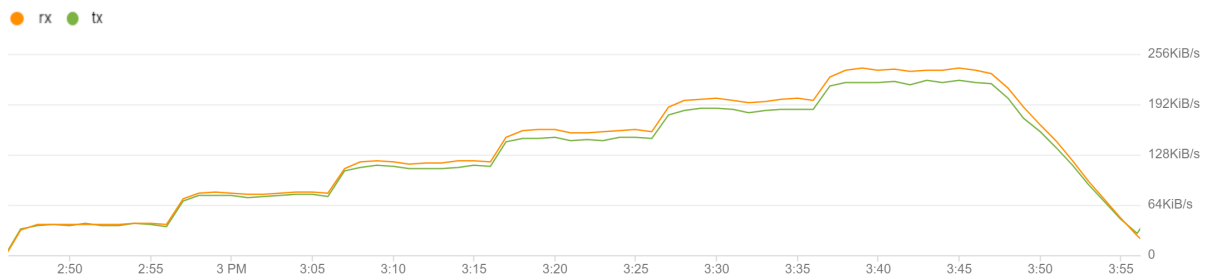


Figure 19 – Network Throughput Consumption - Monitored by StackDriver

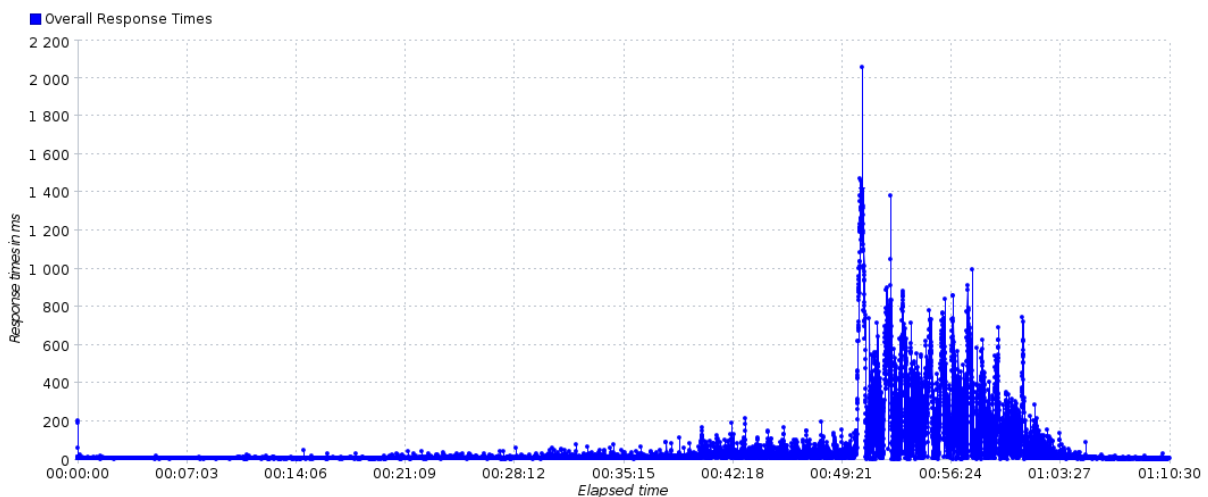


6.5.5 Response Time

The response time measures the server response of every single transaction. It starts when a VU sends a request and ends at the time that the application states that the request has completed.

During the performance of the load tests, the response times for each test were monitored. With PerfMoon monitoring tool, the average response time from the server running the Web application was 164.2ms. PerfMoon's detected that the highest response time was 2084ms at 51 minutes of test execution. With the StackDriver tool, the average response time was 160.6ms and the highest detected value was 1840ms. This response peak delay was also detected during the period when virtual user generation peaked. Figure 20 and Figure 21 graph the monitoring results with PerfMoon and StackDriver respectively.

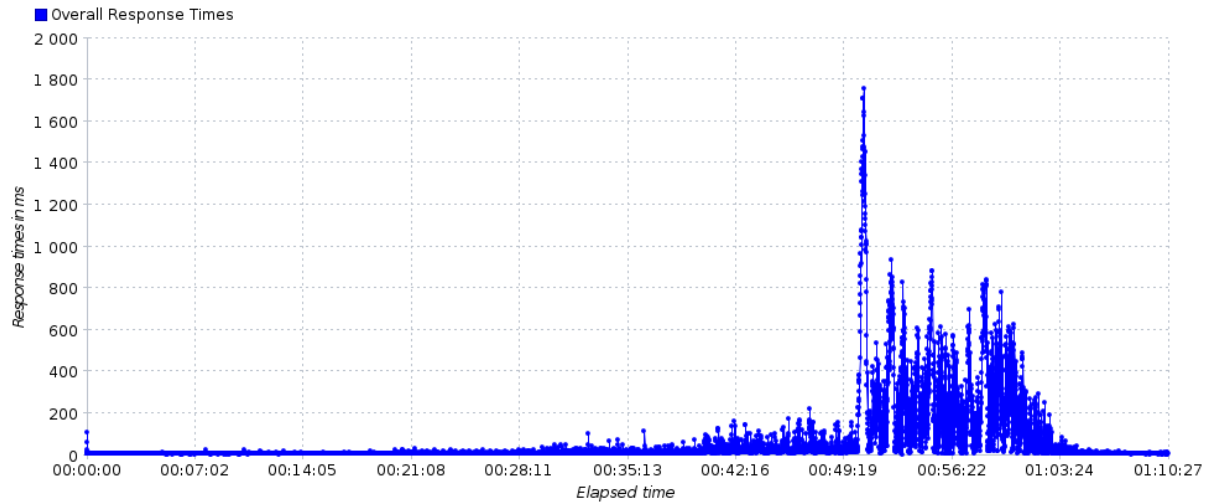
Figure 20 – Response Time - Monitored by PerfMoon



6.5.6 Deviation Results

The deviation formula given in section 6.2 was also applied to the various results generated by the monitoring tools. Table 13 shows the correlation of the results monitored

Figure 21 – Response Time - Monitored by StackDriver



by the treatments PerfMoon and StackDriver tools. In the table is possible to see that the deviation was small and that the PerfMoon tool developed in this work has a degree of precision similar to the commercial tool StackDriver from the company Google.

For the calculation of deviations, only the general averages of log results were used. If the monitoring result is closer to numeral 1, smaller is the deviation and mathematically prove that the monitoring results are similar. The most oscillating metric after applying the deviation calculation was ram memory. The PerfMoon tool consumed an average of 9.5% during testing, while the compared tool used 5.1%. The least oscillating metric with the best deviation results was the response time. PerfMoon's detected an average of 164ms while StackDriver tool 160.6ms. The result of the deviation calculation was 0.951.

Table 13 – Cosine Application.

Correlation	CPU	Memory	Disk I	Disk O	Net. Rx	Net. Tx	Response
CPU	0.922						
Memory		0.772					
Disk I			0.934				
Disk O				0.901			
Net. Rx					0.889		
Net. Tx						0.915	
Response							0.951

6.6 Threats to Validity

In this section, different validity threats related to the case study and experiment are discussed. The author used Creswell e Creswell (2017) to explain different validity threats in the research.

6.6.1 Internal Validity

Internal validity is how well an experiment reduces its own systematic error within the circumstances of the experiment being performed (STILL, 2011).

Internal validity focus on how sure we can be that the treatment actually caused the outcome. There can be other reasons that have caused the result on which we do not have control over or have not measure (FELDT; MAGAZINIUS, 2010).

The internal validity threats in this research are:

- The author never had used the selected monitoring tool (Google StackDriver) once before. To overcome this threat, the author learned how to conduct the performance monitoring by taking help from the online tutorials. After the author learned how to properly use the tool, the experiment was conducted;
- There was a threat that the monitored metrics in the experiment can really explain the outcome the author wants to research. To overcome this threat, some pilot tests were conducted before the execution of the real experiment. The researcher used both tools and tested the scenarios to validate the results of both.

6.6.2 External Validity

External validity is related with whether the results can be generalized outside the scope of the study (FELDT; MAGAZINIUS, 2010).

- There was the threat of network infrastructure with unstable internet speed. This experiment was done in a rented server of Google Cloud Platform where the internet speed was stable and high enough that it would not limit the testing capabilities of any of the tools.

6.6.3 Construct Validity

Construct validity motivation is on the relation between the theory behind the experiment and the interpretations. The interpreted result might not correspond to the effect what is being measured (FELDT; MAGAZINIUS, 2010).

- There was a threat that the selected tool can answer the selected parameters for the experiment. To mitigate this threat, different literature, through the systematic mapping previously discussed in this term paper, and the official websites of the selected tools were studied and confirmed that the tool can satisfy the selected parameters.

6.6.4 Conclusion Validity

Conclusion validity concentrate on how sure the treatment used in an experiment really is related to the actual result obtained (FELDT; MAGAZINIUS, 2010).

- Conclusion validity is a threat that can lead the research to an incorrect conclusion. To mitigate this threat, the author acquired a background in performance testing research and was assisted by his supervisor and co-supervisor, both researchers in the field of software engineering and performance testing;
- The author used a web application for testing. There was a threat that if the web application is down from hosting side. To overcome this threat, the experiment was conducted in a proprietary solution, which was hosted in a server the researcher had total control of.

6.7 Chapter Summary

This chapter describes how the experimental benchmarking planning and execution decisions were made.

7 CONCLUSIONS AND FUTURE WORK

In this chapter the conclusions and future works regarding this work are presented.

7.1 Conclusions

Through this study, it was possible to observe the importance of the process of software testing. Its process is part of the software development cycle and has as main objective to reveal faults or bugs so that they are corrected until the final product reaches the expected quality.

Performance tests consist of evaluating the responsiveness, robustness, availability, reliability and scalability of an application, according to the number of concurrent connections, evaluating its performance at high workload and considering its behavior under normal circumstances. In particular, the purpose of such experiments may be to ensure that the software does not present problems or unavailability in conditions of insufficient computing resources when working in high competition or suffering some attack of negation of service.

Professional software testers (also called test analysts, test engineers, etc.) are accustomed to perform tests of different natures and purposes, involving not only the functional tests of the application, but several other activities such as: evaluation requirements specification, technical project evaluation, checks on other documents, performance and capacity tests, and interface evaluation.

With the results of this study, it is noticed that one of the great problems found in the area of performance and monitoring test is the fact that the performance specialists need not only to design the tests, but also to be aware of the technologies that will be used to execute them. It can become a problem if the tester does not have this knowledge. It is a common problem that can occur and affect the project in several ways, with the most significant impact on the final cost of the project. Tests that run erroneously or resources allocated and distributed incorrectly cause losses and delays.

Software monitoring has been explored for a long time in various fields, such as performance evaluation and enhancement, fault tolerance, and standalone computing. Based on a systematic literature mapping, which analyzed and summarized research related to traditional software monitoring, it was possible to propose a taxonomy (can be seen in Figure 4.3 that will serve as a catalog for classification of monitoring tools, assisting professionals and students in the process of identifying the characteristics of these tools. This study also proposes a tool called PerfMoon (see in Chapter 5). The tool is proposed as a module of a group of tools that, when integrated, can fill and assist current gaps found in the literature such as modeling test scripts from models and performance and monitoring tests.

The results of experimental benchmarking indicate that the tool proposed and developed in this paper, PerfMoon, has a level of accuracy similar to the commercial

StackDriver tool owned by one of the world's largest technology companies, Google Company.

For future work PerfMoon tool needs to be integrated with the remaining tool modules of the COSMOS performance testing solution. The tool also lacks more functionality so it can span more web servers and not just Apache Web Server.

The description of all the tasks performed in this term paper is presented in section 7.4 in the form of a schedule. The main lessons learned are described in the following section.

7.2 Lessons Learned

During the preparation of this term paper, the analysis of results revealed common patterns that have resulted in some lessons learned. For each lesson, this section presents support from the case studies and the implications of that lesson for software engineering.

7.2.1 Efficient Systematic Mapping Protocol

Protocols set out the plans for the conduct of systematic mappings. These protocols describes the methods used in the mapping, synthesizes and narratively presents the studies collected and the data they contain. The creation of these protocols is a complex activity, because it requires a lot of time for reviews and improvements, and requires that researchers have previous experience in systematic studies.

7.2.2 Testing Tools Classification

In the literature software testing and monitoring tools are classified in different ways based on different criteria, such as complexity, development technology or intended use. This process is complex and time-consuming, as this information is sometimes not explicit in primary studies.

7.2.3 Software Testing

Software testing are really required to point out the errors and failures that were been committed during the development phases. These tests guarantee the high quality of the software product making it reliable.

7.2.4 Performance Testing

Performance testing are essential, but they must follow a correct execution structure. Testing with failed results can lead to wrong decisions in the project and this can raise the cost of the project. There are different practices for performing performance tests, these should be chosen according to the specification of the test to be performed.

7.2.5 Performance Monitoring

Performance monitoring provide data that can quickly isolate and resolve problems that can adversely affect software performance. These tests monitor several metrics of a given software over a period of time and generate results for analysis.

7.3 Publications

It is worth mentioning successful publications, attempted publications, and planning for future publications derived from this term paper. The events described below were sorted in chronological order.

- **ESEM 2019** - The ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) is the premier conference for presenting research results related to empirical software engineering. (Attempted publication.)
- **SBES 2019** - The XXXIII Brazilian Symposium on Software Engineering (SBES), annually promoted by the Brazilian Computer Society (SBC), is the premier Software Engineering event in Latin America. SBES is held in conjunction with CBSOFT - Brazilian Conference on Software: Theory and Practice. (Attempted publication.)
- **IPCCC 2019** - The International Performance, Computing, and Communications Conference is a premier IEEE conference presenting research in the performance of computer and communication systems. For over three-and-a-half decades, IPCCC has been a research forum for academic, industrial, and government researchers. (Paper was accepted, but unfortunately it was not possible to gather resources to travel to London.)
- **ERES 2019** - The Regional School of Software Engineering (ERES) is an event promoted annually by the Brazilian Computer Society (SBC). The third edition of the event, ERES 2019, has taken place in Rio do Sul (SC), the Alto Vale do Itajaí region, from October 7 to 9, 2019, and was jointly organized by the Federal Institute of Santa Catarina (IFC) and Santa Catarina State University (UDESC). (Paper accepted and presented.)
- **SIEPE 2019** - The 11th International Teaching, Research and Extension Salon (SIEPE) was held in Santana do Livramento (Brazil) and Rivera (Uruguay) on October 22, 23 and 24, 2019. (Paper accepted and presented in oral and poster modalities.)
- **SAC 2020** - The 35th ACM/SIGAPP Symposium On Applied Computing (SAC). In the Software Verification and Testing (SVT) Track. (Currently under review.)

- **STVR** - Software Testing, Verification and Reliability (STVR) is an international journal, publishing 8 issues per year. It publishes papers on theoretical and practical issues of software testing, verification and reliability. (Planned for publication.)

7.4 Schedule

In this section, the task schedule is presented and the performed tasks for completing this study are also defined. The Table 14 lists these tasks and the date they were executed.

Term Paper 1: The development of this present study was divided in 4 stages that are detailed below:

- **Planning a SMS:** In this stage, the planning of the SMS was developed. During these 4 months the protocol was developed and refined. This stage was responsible for the definition of research questions, identification of research sources (databases), inclusion and exclusion criteria, data extraction metrics and rules for quality evaluation.
- **Executing the SMS:** Started in December 2018, this step was responsible for conducting the SMS. During this period the process of searching for the primary studies, the application of inclusion and exclusion rules, quality evaluation and the extraction of data according to the metrics defined in the protocol were carried out.
- **Analysis Designing:** The analysis and design of the proposed tool occurred during the period from May to June.
- **Writing Term Paper:** The writing of this term paper occurred in the period from April to June.

Term Paper 2: The study to be developed was divided into 4 stages, they are:

- **Tool Development:** PerfMoon's prototype development will commence in late June.
- **Planning Evaluation:** Determine a general approach and establish an empirical evaluation framework.
- **Conducting Evaluation:** Evaluate the PerfMoon tool according to the indicators proposed in the previous step.
- **Writing Term Paper:** Writing and presentation of the final term paper.

Although this report ends here, PerfMoon's future evolution is already planned to be executed within LESSE's research group in the years to come.

Table 14 – Schedule.

Task	2018/2				2019/1							2019/2			
	Set	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov
Planning the Systematic Mapping					-	-	-	-	-	-	-	-	-	-	-
Executing the Systematic Mapping	-	-	-									-	-	-	-
Writing Term Paper Project	-	-	-	-	-	-	-					-	-	-	-
Analysis & Design	-	-	-	-	-	-	-					-	-	-	-
Presenting Term Paper Project	-	-	-	-	-	-	-	-				-	-	-	-
PerfMoon's Development	-	-	-	-	-	-	-	-	-						
Plan Tool Evaluation	-	-	-	-	-	-	-	-	-	-					
Writing Term Paper	-	-	-	-	-	-	-	-	-	-	-	-	-		
Tool Evaluation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Presenting Term Paper	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
Refining Term Paper for Homologation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

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 43.
- 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 43.
- AHMAD, I. Easy and efficient disk i/o workload characterization in vmware esx server. In: **Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization**. Washington, DC, USA: IEEE Computer Society, 2007. (IISWC '07), p. 149–158. ISBN 978-1-4244-1561-8. Disponível em: <<https://doi.org/10.1109/IISWC.2007.4362191>>. Cited in page 72.
- 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 43.
- AMMANN, P.; OFFUTT, J. **Introduction to software testing**. [S.l.]: Cambridge University Press, 2016. Cited in page 23.
- 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 43.
- 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 37.
- BEIZER, B. **Software System Testing and Quality Assurance**. New York, NY, USA: Van Nostrand Reinhold Co., 1984. ISBN 0-442-21306-9. Cited in page 63.
- BENEDIKT, M.; FREIRE, J.; GODEFROID, P. Veriweb: Automatically testing dynamic web sites. In: CITeseer. **In Proceedings of 11th International World Wide Web Conference (WW W'2002)**. [S.l.], 2002. Cited in page 24.
- BERNARDINO, M.; ZORZO, A. F.; RODRIGUES, E. M. Canopus: A Domain-Specific Language for Modeling Performance Testing. In: **2016 IEEE International Conference on Software Testing, Verification and Validation (ICST'16)**. [S.l.: s.n.], 2016. p. 157–167. Cited in page 57.
- BERTOLINO, A. Software testing research: Achievements, challenges, dreams. In: IEEE COMPUTER SOCIETY. **2007 Future of Software Engineering**. [S.l.], 2007. Cited in page 23.
- 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 43.

- CHAWLA, D.; SODHI, N. **Research methodology: Concepts and cases**. [S.l.]: Vikas Publishing House, 2011. Cited in page 27.
- CHEUNG, C. M. K.; LEE, M. K. O. The asymmetric effect of website attribute performance on satisfaction: An empirical study. **Proceedings of the 38th Annual Hawaii International Conference on System Sciences**, p. 175c–175c, 2005. Cited in page 24.
- 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 43.
- 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 48.
- COOK, T.; CAMPBELL, D. **Quasi-Experimentation: Design and Analysis Issues for Field Settings**. [S.l.]: Houghton Mifflin, 1979. Cited in page 54.
- COSTA, V. Proposal of a Tool to Generate Workloads on Web Applications - LoadSun. **forthcoming**, 2019. Cited in page 64.
- CRESWELL, J. W.; CRESWELL, J. D. **Research design: Qualitative, quantitative, and mixed methods approaches**. [S.l.]: Sage publications, 2017. Cited in page 75.
- 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 43.
- 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 44 and 50.
- DEGWEKAR, S.; SU, S. Y. W.; LAM, H. Constraint specification and processing in web services publication and discovery. In: **Proceedings. IEEE International Conference on Web Services, 2004**. [S.l.: s.n.], 2004. p. 210–217. Cited in page 23.
- 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 43.
- 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 43.
- EHLERS, J. **Self-adaptive performance monitoring for component-based software systems**. Tese (Doutorado) — Christian-Albrechts Universität Kiel, 2012. Cited in page 32.

EICHELBERGER, H. et al. Adaptive application performance management for big data stream processing. **Softwaretechnik Trends**, v. 35, n. 3, p. 35–37, 2015. Cited 2 times in pages 34 and 35.

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 37, 54, and 55.

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 43.

FELDERER, M.; SCHIEFERDECKER, I. A taxonomy of risk-based testing. **Int. J. Softw. Tools Technol. Transf.**, Springer-Verlag, Berlin, Heidelberg, v. 16, n. 5, p. 559–568, out. 2014. ISSN 1433-2779. Disponível em: <<http://dx.doi.org/10.1007/s10009-014-0332-3>>. Cited in page 31.

FELDT, R.; MAGAZINIUS, A. Validity threats in empirical software engineering research—an initial survey. In: **Seke**. [S.l.: s.n.], 2010. p. 374–379. Cited 2 times in pages 76 and 77.

FOSTER, I. **Designing and building parallel programs: concepts and tools for parallel software engineering**. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1995. Cited in page 32.

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 43.

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 43.

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 55.

JAMIL, M. A. et al. Software testing techniques: A literature review. In: . [S.l.: s.n.], 2016. p. 177–182. Cited in page 31.

JANES, A.; LENARDUZZI, V.; STAN, A. C. A continuous software quality monitoring approach for small and medium enterprises. In: **Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion**. New York, NY, USA: ACM, 2017. (ICPE '17 Companion), p. 97–100. ISBN 978-1-4503-4899-7. Disponível em: <<http://doi.acm.org/10.1145/3053600.3053618>>. Cited in page 34.

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 54 and 55.

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 43.

- 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 43.
- 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 43.
- KHAN, M. E. Different forms of software testing techniques for finding errors. **International Journal of Computer Science Issues (IJCSI)**, Citeseer, v. 7, n. 3, p. 24, 2010. Cited in page 32.
- 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 43.
- 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 43.
- 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 43.
- 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 43.
- 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 38.
- KOREL, B. Automated software test data generation. **IEEE Trans. Softw. Eng.**, IEEE Press, Piscataway, NJ, USA, v. 16, n. 8, p. 870–879, ago. 1990. ISSN 0098-5589. Disponível em: <<http://dx.doi.org/10.1109/32.57624>>. Cited in page 31.
- KOZIOLEK, H. Goal, question, metric. In: **Dependability metrics**. [S.l.]: Springer, 2008. p. 39–42. Cited in page 37.
- 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 43.
- 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 43.
- 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 45.

KUMAR, R. **Research Methodology: An Introduction**. [S.l.]: New Dehli: APH Publishing Corporation, 2008. Cited in page 27.

LAMANNA, D. D.; SKENE, J.; EMMERICH, W. Slang: A language for defining service level agreements. In: **IEEE COMPUTER SOC. NINTH IEEE WORKSHOP ON FUTURE TRENDS OF DISTRIBUTED COMPUTING SYSTEMS, PROCEEDINGS**. [S.l.], 2003. p. 100–106. Cited in page 34.

LANGDON, C. S. The state of web services. **Computer**, v. 36, n. 7, p. 93–94, July 2003. ISSN 0018-9162. Cited in page 23.

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 53.

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 43.

LIU, X.; IFTIKHAR, N.; XIE, X. Survey of real-time processing systems for big data. In: **Proceedings of the 18th International Database Engineering & Applications Symposium**. New York, NY, USA: ACM, 2014. (IDEAS '14), p. 356–361. ISBN 978-1-4503-2627-8. Disponível em: <<http://doi.acm.org/10.1145/2628194.2628251>>. Cited in page 69.

MATHUR, A. P. Performance, effectiveness, and reliability issues in software testing. In: IEEE. **[1991] Proceedings The Fifteenth Annual International Computer Software & Applications Conference**. [S.l.], 1991. p. 604–605. Cited in page 32.

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 43.

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 44 and 48.

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 43.

MYERS, G. J.; SANDLER, C. **The Art of Software Testing**. USA: John Wiley & Sons, 2004. ISBN 0471469122. Cited in page 23.

NEIL, N. J. **Learning Ubuntu 14.04 LTS: A Beginners Guide to Linux**. 2nd. ed. USA: CreateSpace Independent Publishing Platform, 2016. ISBN 1539852598, 9781539852599. Cited in page 69.

- 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 43.
- PATTON, R. **Software Testing (2Nd Edition)**. Indianapolis, IN, USA: Sams, 2005. ISBN 0672327988. Cited in page 31.
- PERRY, W. E. **Effective methods for software testing: Includes complete guidelines, Checklists, and Templates**. [S.l.]: John Wiley & Sons, 2007. Cited in page 23.
- PODELKO, A. Reinventing performance testing. In: CMG. [S.l.], 2016. Cited in page 43.
- 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 43.
- 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 43.
- 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 43.
- 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 43.
- SAHAI, A. et al. Automated sla monitoring for web services. In: SPRINGER. **International Workshop on Distributed Systems: Operations and Management**. [S.l.], 2002. p. 28–41. Cited in page 34.
- SCHROEDER, B. A. On-line monitoring: A tutorial. **Computer**, IEEE, v. 28, n. 6, p. 72–78, 1995. Cited in page 33.
- SHAMS, M.; KRISHNAMURTHY, D.; FAR, B. A model-based approach for testing the performance of web applications. In: **Proc. ACM 3rd international workshop on Software quality assurance**. [S.l.: s.n.], 2006. p. 54–61. Cited in page 24.
- 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 55.
- 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 43.

SMITH, C. U. **Performance Engineering of Software Systems**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990. ISBN 0201537699. Cited 2 times in pages 31 and 33.

SMITH, C. U.; WILLIAMS, L. G. Software performance engineering. In: **UML for Real**. [S.l.]: Springer, 2003. p. 343–365. Cited in page 31.

STILL, J. D. Experimental design: Does external validity trump internal validity? **interactions**, ACM, New York, NY, USA, v. 18, n. 3, p. 66–68, maio 2011. ISSN 1072-5520. Disponível em: <<http://doi.acm.org/10.1145/1962438.1962453>>. Cited in page 76.

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 43.

SUBRAYA, B.; SUBRAHMANYA, S. Object driven performance testing of web applications. In: IEEE. **Proceedings First Asia-Pacific Conference on Quality Software**. [S.l.], 2000. p. 17–26. Cited in page 35.

VASWANI, V. **PHP: A BEGINNER'S GUIDE**. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 2009. ISBN 0071549013, 9780071549011. Cited in page 69.

WAZLAWICK, R. **Metodologia de pesquisa para ciência da computação**. [S.l.]: Elsevier Brasil, 2017. v. 2. Cited in page 27.

WEYUKER, E. J.; VOKOLOS, F. I. Experience with performance testing of software systems: issues, an approach, and case study. **IEEE transactions on software engineering**, IEEE, v. 26, n. 12, p. 1147–1156, 2000. Cited in page 23.

WHOLEY, J. S.; HATRY, H. P. The case for performance monitoring. **Public administration review**, American Society for Public Administration, v. 52, n. 6, p. 604, 1992. Cited in page 34.

WOODSIDE, M.; FRANKS, G.; PETRIU, D. C. The future of software performance engineering. In: IEEE COMPUTER SOCIETY. **2007 Future of Software Engineering**. [S.l.], 2007. p. 171–187. Cited in page 31.

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 43.

YAN, M. et al. Delivering Web service load testing as a service with a global cloud. v. 27, n. 3, 2014. Cited in page 43.

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 43.

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 43.

- 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 43.
- YIN, R.; SAGE. **Case Study Research: Design and Methods**. SAGE Publications, 2003. (Applied Social Research Methods). ISBN 9780761925521. Disponível em: <https://books.google.com.br/books?id=BWea_9ZGQMwC>. Cited in page 63.
- 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 43.
- ZHENG, P. et al. A detailed and real-time performance monitoring framework for blockchain systems. In: **Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice**. New York, NY, USA: ACM, 2018. (ICSE-SEIP '18), p. 134–143. ISBN 978-1-4503-5659-6. Disponível em: <<http://doi.acm.org/10.1145/3183519.3183546>>. Cited in page 34.
- 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 43.

Index

BPMN, 28

CNPQ, 25

CR, 19, 48

DdS, 63

DoS, 63

EC, 39

FR, 20, 58

IC, 39

LESSE, 25, 57, 60, 82

MBT, 19, 50

NFR, 20, 59

QoS, 34

RQ, 19, 38

SBS, 34

SE, 23

SLA, 35

SMS, 25, 37, 42, 44, 45, 54, 82

SPE, 31, 32

SUT, 20, 65, 66

UNIPAMPA, 57

, 66, 67

VU, 51, 67, 68, 74

WBS, 34