FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Performance Testing in Heterogeneous and Distributed Engineering Environments

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Christian Macho
Matrikelnummer 0726108

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffl
Mitwirkung: Dipl.-Ing. Dietmar Winkler

Wien, 21.1.2015

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Performance Testing in Heterogeneous and Distributed Engineering Environments

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Christian Macho
Registration Number 0726108

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffl
Assistance: Dipl.-Ing. Dietmar Winkler

Vienna, 21.1.2015

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Christian Macho
Walter Klenner Straße 3, 3830 Waidhofen an der Thaya

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

Software Quality is an important aspect in modern Software Engineering and Software Engineering development processes. To ensure the quality of modern software systems, there exist many types of tests and approaches, such as acceptance testing or performance testing. As Software Engineering is a growing field and methods of Software Engineering are also transferred to other disciplines, this transfer is also interesting for Software Quality methods. One of the disciplines that Software Engineering methods are heading to, is the area of Automation Systems as these systems get larger and more complex. There are approaches to solve the occurring challenges such as the Automation Service Bus (ASB). Due to the raising complexity, a need of controlling the quality also raises. At that point Software Testing approaches can be applied. Performance Testing is a part of Software Testing and should therefore also be taken into account.

Performance Testing is a difficult task. There are many available tools and they all have different features and aims. Therefore, there is a challenge to determine a representative list of criteria that reflect the needs of the users concerning the usage of a tool. Subsequently, a tool has to be found that fulfills the criteria best. In order to be able to execute performance tests, a method outlining how to do performance tests in this environment has to be found and summarized in a framework. Further, this framework has to be shown working.

For this purpose a tool study of available Performance Testing tools is performed with a list of collected requirements of the users. As a result of this tools study, a most suitable tool is found. Subsequently, a Performance Testing Framework is designed and implemented with the most suitable tool. This framework is then applied on a web front-end of an industrial application running in the Automation Service environment. It is checked if the framework is feasible for applying it in the research environment.

The results of this thesis present a list of criteria, tailored to the research environment, to find a tool that is able to allow a Performance Analysis - JMeter. Around and with JMeter a framework is implemented and the case study shows that the framework with JMeter works well for this use case. There is space for future study, such as extending the tests or distribute the Performance Analysis.

Kurzfassung

Software Qualität ist ein wichtiger Aspekt im modernen Software Engineering und in Software Engineering Prozessen. Um die Qualität von modernen Software Systemen zu sichern, gibt es viele Arten von Tests und Ansätzen, wie zum Beispiel Akzeptanz Tests oder Performance Tests. Da Software Engineering ein wachsendes Gebiet ist und die Methoden des Software Engineering auch auf andere Disziplinen übertragen werden, ist es auch interessant, Software Qualitätssicherungsmethoden auf andere Disziplinen zu übertragen. Eine solcher Disziplinen, auf die Software Engineering Methoden angewandt werden, sind Automations Systeme, die ebenfalls immer größer und komplexer werden. Es gibt Ansätze die auftretenden Herausforderungen zu lösen, wie zum Beispiel den Automation Service Bus (ASB). Aufgrund der steigenden Komplexität, entsteht auch das Bedürfnis die Qualität zu kontrollieren. An diesem Punkt können Software Testing Ansätze angewandt werden. Performance Testing ist Teil des Software Testing und sollte deshalb auch in Betracht gezogen werden

Performance Testing ist eine schwierige Aufgabe. Es gibt viele verfügbare Tools, die alle verschiedene Funktionen und Ziele haben. Deshalb ist es eine Herausforderung eine repräsentative Liste an Kriterien zu ermitteln, die den Bedürfnissen der User an das Tool entspricht. Im Anschluss muss ein Tool gefunden werden, dass die Kriterien am besten erfüllt. Um Performance Tests ausführen zu können, müssen Methoden gefunden werden, wie Performance Tests in der Umgebung am besten ausgeführt werden können. Diese Methoden werden in ein Framework zusammen gefasst. Weiters muss gezeigt werden, dass dieses Framework funktioniert.

Zu diesem Zweck wird eine Tool Studie, mit einer Liste an Anforderungen der User, durchgeführt. Ein Ergebnis der Studie ist ein Tool, welches die Anforderungen am besten erfüllt. Danach wird ein Performance Testing Framework entworfen und umgesetzt mit dem vielversprechendsten Tool. Dieses Framework wird dann auf ein Web Front-End einer Industrie Applikation angewandt, die in einem Automation Environment läuft. Weiters wird überprüft ob das Framework im Rahmen der Forschungsumgebung realisierbar und anwendbar ist.

Die Ergebnisse dieser Arbeit präsentieren eine Liste an Kriterien, die auf die Forschungsumgebung zugeschnitten sind, um ein Tool zu finden, dass es eine Performance Analyse möglich macht - JMeter. Um und mit JMeter wird ein Framework umgesetzt und eine Fallstudie zeigt dass das Framework mit JMeter in diesem Use Case gut funktioniert. Es gibt noch Platz für weitere Studien, wie zum Beispiel die Tests zu erweitern oder die Performance Analyse zu verteilen.

Contents

CHAPTER 1

Introduction

This chapter introduces into the area of research and presents a motivation for the thesis. It further expresses the problem that is addressed and the expected results as aim of the thesis. The chapter closes with an overview of the structure of the thesis.

1.1 Motivation

Ensuring the quality of nowadays applications is a very important part in modern Software Engineering. More and more companies have introduced quality assurance methods to their development processes. As quality assurance is a wide spread area, each of the parts may be important to the project to help improving the quality. Software Performance Testing is a part of the quality assuring methods. Hence, it is also important for projects. Further Software Engineering processes and Software Engineering itself spreads more over to other application areas and therefore gets involved in interdisciplinary projects.

One of these new environments in which Software Engineering raises, is the area of Automation Systems. In this area, production systems get more complex. As a consequence there is a need for an easier handling of those systems. There already exists research for integrating the technical parts of Automation Systems [2, 24, 49, 75] as well as for the general integration of Software Engineering tools and systems called the "Automation Service Bus" (ASB) [19]. The raising complexity of Automation Systems also requests a process that is able to check the quality of the product. At this point Software Testing comes into play. There are already some approaches that deal with the methods of Software Testing and try to apply them to the area of Automation Systems [45, 86–88]. In these publications there is no explicit work that deals with Performance Testing in this field. Hence, this thesis focuses on the application of Performance Testing methods from the typical Software Engineering/Software Testing field in the area of Automation Systems.

The addressed audience by this work are engineers in the Automation Systems area and people developing processes for this area. They have to test their systems and ensure the quality

of the product and hence have to steadily improve the used methods to secure the validity and correctness of the results.

1.2 Problem Statement

As seen in section 1.1, there is ongoing research in terms of applying processes of modern Software Engineering to the discipline of heterogeneous distributed systems, such as the Automation environment provides. As well for common Software Engineering processes the difficulty is also present when applying Software Testing processes. There are some problems that arise when adopting Software Testing processes to Automation environments [86–88]. Some of the aspects of Software Testing have already been applied to the Automation environment. As in most disciplines, performance plays a remarkable role also in Automation environments. The applications have to provide a satisfying performance in means of response time, throughput or scalability. At this point the engineers are confronted with the task of testing the requirements for performance issues. The recent research has shown that Software Engineering processes can be applied to Automation Systems area [86–88] especially for testing. In contrast to unit testing there is no research that has dealt with the application of software performance testing processes in the Automation Systems discipline. This thesis seizes this gap, evaluates tool candidates for applying performance analysis and develops a solution approach for heterogeneous distributed systems in an Automation environment in means of Performance Testing.

Connected to that we address three tasks. All three are directly correlated with the ASB. Figure 1.1 shows the location of the problems concerning the (extracted) research environment.

Number 1 represents the selection of the tools. This includes the problem of how to select a tool which further includes how to select values representing a good choice. We want to find a tool that is most promising concerning the parameters chosen in this part. The outcome is a tool (or a set of tools) that fulfills the requirements best. Subsequently there is a need of a framework that uses this tool (or set of tools) to do performance analysis. This is represented by number 2 in figure 1.1. We design a framework around the most promising tool from number 1 and implement the framework. Addressing number 3, we use the designed and implemented framework of number 2 and perform a case study on an existing application in the research environment. For that purpose, we use a prototype of a web application front end of an industry partner. We perform a performance analysis and check whether the implemented framework is feasible and applicable in our research environment.

1.3 Motivating Example

As seen in the previous paragraphs, there is research for and around the ASB. We now introduce a motivating example for this thesis. In the area of the ASB many applications can be deployed and can communicate with the ASB. Therefore, performance is an important aspect in this environment. The experts who deal with those applications do not want do waste time with waiting for the ASB or any connected tool. To ensure that the requirements stated are met, there has to be a quality assuring instance that proves the required performance is met.

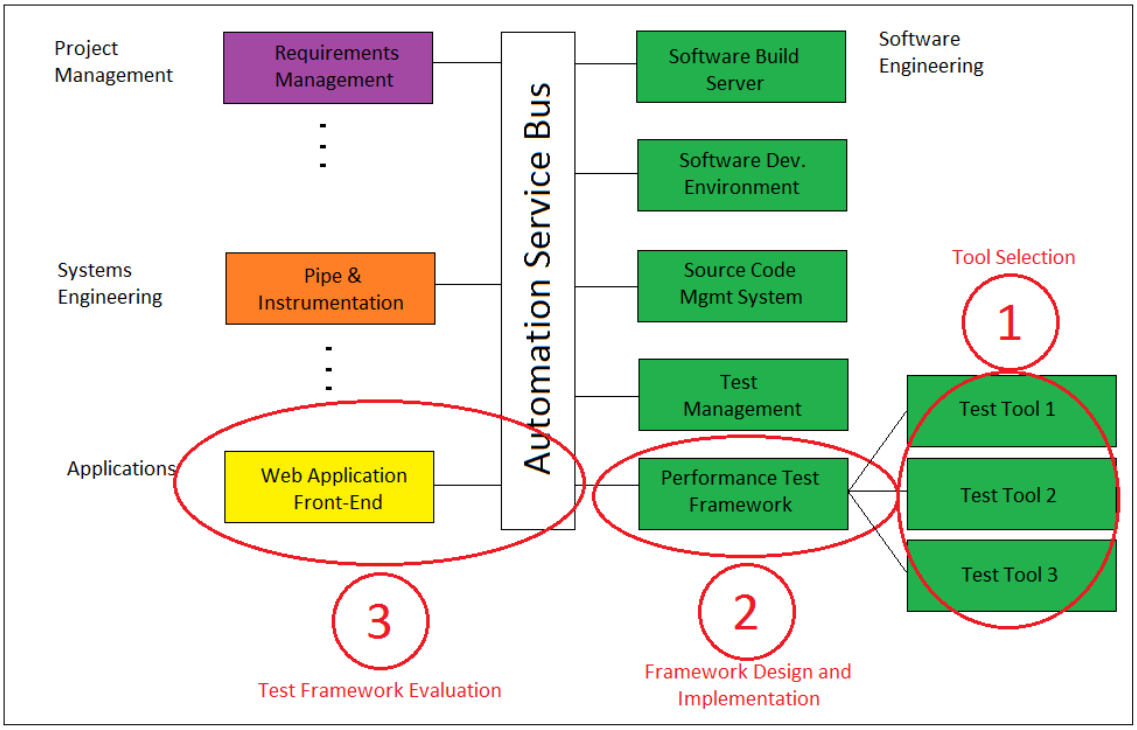


Figure 1.1: Illustration of the location of the research problems in the (extracted) environment

Especially web applications are very sensible in concerns of performance. Users expect to immediately get a response and see the pages displayed. They loose interest and leave or even worse they do not use the application any more. Therefore, applications have to be performance tested before they can be released. On the side of the ASB, there also exists a web application for managing tasks, such as user management or project management. As well as these, there also exist further features such as the checkin feature which provides the functionality of checking in signal data files to the ASB. This feature is time consuming and has to be tested in means of performance and so has the rest of the application.

For all these performance testing needs a tool has to be found that suffices the needs of the surrounding environment in which the application is run. With the chosen tool or set of tools performance testing can be established in the ASB area.

1.4 Aim of the Work

The aim of this thesis is to establish performance testing with a fitting tool. This overall aim splits up into three main parts.

1.4.1 Tool Selection

The first part aims to gain knowledge about the research environment in means of performance testing issues. The requirements that are needed to be able to perform a tool evaluation are determined and weighted in means of importance for the tool. The outcome of this part is an evaluation sheet that can be applied to perform a tool evaluation for the heterogeneous automation environment.

Further, we list all available tools and prepare for the evaluation study. We perform the evaluation based on the criteria selection of part one with the listed tools. The outcome of this part is a ranking of tools and especially winning tool (or a set of tools) that fits best for performance testing in the sense of performance testing.

1.4.2 Test Framework Design and Implementation

The second part is the design and implementation of a performance testing process with the winning tool (or set of tools). The outcome of this part is a runnable prototype that can be applied to the research environment.

1.4.3 Test Framework Evaluation

The third and last part is a case study in which we apply the our framework to an existing project in the research area and show that the approach is feasible. The outcome here is a statement for the framework's functionality.

1.5 Methodological Approach

The methodological approach is split into two parts. The evaluation part uses an approach of Robert M. Poston and Michael P. Sexton [74] for the process of evaluation as base. The approach is adopted where necessary. Therefore, an evaluation sheet is created for a survey to identify the needs of the target users. The features and properties of each tool are then evaluated. As a last step, the evaluated tools are taken into account and in order to the users' wishes a tool-value is calculated with the weight gained out of the users survey. The evaluation then presents a most suitable tool which further is used to implement the performance tests in the Automation Systems environment.

In the second part of the thesis we use a design approach to describe the target framework which is the implemented as a prototype. We present a design of the framework and describe the needed tests in order to be able to perform a performance analysis. Then a prototype implementation is made based on this framework design.

Further, we use a case study to evaluate whether the framework implementation performs well in our research environment and to see how the results of the tested application are.

1.6 Structure of the Work

The structure of the work is illustrated in figure 1.2 and mapped to the relating parts of the work. Further, we describe each part as follows.

After this chapter the related work is presented in chapter 2. This includes the related work concerning performance testing and the research environment. The related work chapter closes with the description of the methodological approach.

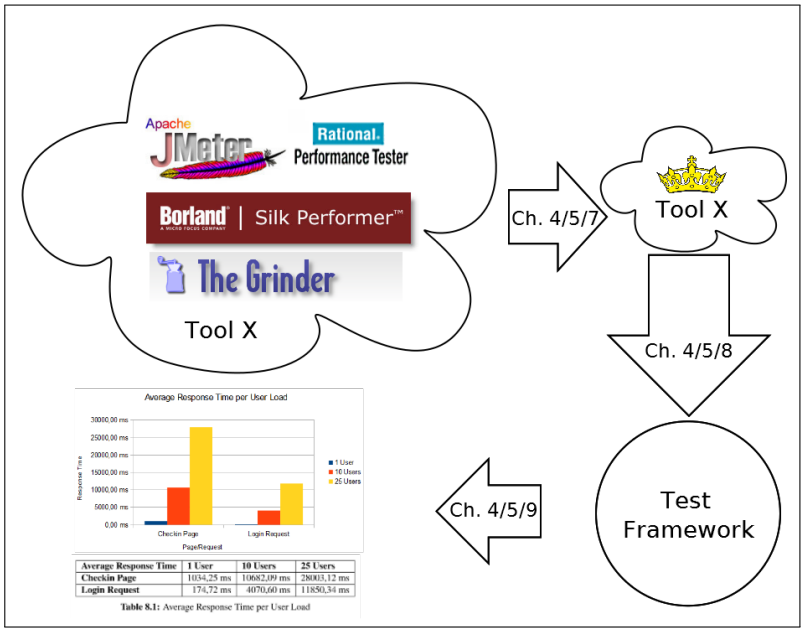
Chapter 3 gives a detailed insight of the research environment the thesis is moving in and describes an use case that is treated throughout the rest of the work. It further expresses the research issues that are addressed and explains the contribution of this thesis.

This is followed by chapter 4 which basically contains the way this thesis is doing the research. At first, a tool evaluation process is described how a suitable tool is found. Then, the design of the implementation is discussed as last point of this chapter.

Next, chapter 5 presents the results of the tool evaluation study and shows the winning or winning set of tools that is used for the remainder of the study. Chapter 6 presents the results of the implemented framework. It explains the implementation and its properties. Chapter 7 gives and detailed view on the results of the case study which is made to show that the chosen approach and framework is feasible for the research environment.

Chapter 8 expresses the limitations of the implemented work and encourages for discussion.

The thesis closes with chapter 9 in which the work is reconsidered and the research questions is answered in order to the evaluated data.



Average Response Time	1 User	10 Users	25 Users
Checkin Page	1034,25 ms	10682,09 ms	28003,12 ms
Login Request	174,72 ms	4070,60 ms	11850,34 ms

Table 8.1: Average Response Time per User Load

Figure 1.2: Illustration of the structure of the work

CHAPTER 2

Related Work

In this chapter the related work for this theses is presented. We start with an introduction to Heterogeneous and Distributed Engineering (Projects) in section 2.1 Then we continue with an overview over performance testing in section 2.2. In section 2.3.1 the methodological approach for the tool evaluation is presented. This chapter closes with a conclusion of the related work in section 2.4

2.1 Heterogeneous and Distributed Engineering Projects

In modern Software Engineering distributed systems become more and more important. Thus, the methods of distributed systems are extended and new methods are developed. In this chapter we present a short introduction to the state of the art methods and approaches and link it with today's heterogeneous distributed engineering.

2.1.1 Types of Distributed Engineering

In this section we present the basic ideas of how to make a distributed environment in a software project. Most of these approaches are based on Integration Patterns that are presented in [49] We introduce the concepts of message-oriented middleware (MoM), Service Oriented Architecture (SOA) and Bus-Systems to get an overview of the state of the art approaches. There are many other approaches, such as CORBA [43], RMI [20] or Space Based Computing [37]. However, we focus on the approaches relevant for the remainder of the thesis.

Message-oriented Middleware

This approach is based on an idea that every kind of communication is done via so called messages. Messages can have different types and payloads. This concept assumes that there is a messaging server which receives the messages and distributes them to the targeted clients. Figure 2.1 gives an overview of how messaging systems work.

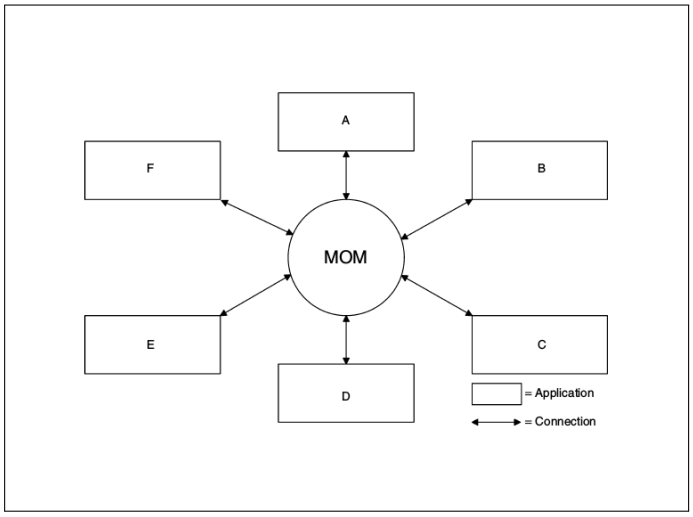


Figure 2.1: Sketch of Message-oriented Middleware systems from [28]

Message-oriented Middleware provides communication models, such as publish/subscribe or point-to-point communication [28, 64]. The biggest benefit of Message-oriented Middleware concepts is the abstraction of the receiving and sending actions. You simply announce what you want to send/receive and where to, and the rest is done by the MoM.

Service Oriented Architecture

Another approach for implementing a distributed environment is SOA [50, 66]. This approach works as figure 2.2 shows.

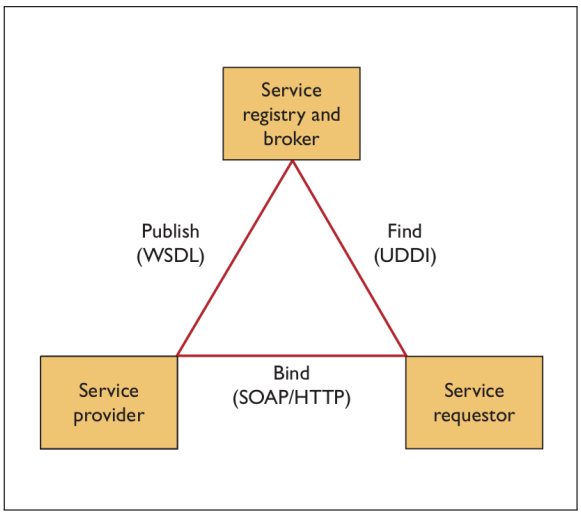


Figure 2.2: Schematic overview of SOA from [50]

There are three parts that are involved in a SOA environment. A Service Provider publishes its service via WSDL to the Service Registry and the Service requestor finds a service via the Service Registry. The communication afterwards is processed directly between the requestor and the provider. This brings the effect of a completely loosely coupled system which is a main benefit of SOA based systems.

Enterprise Service Bus

The Enterprise Service Bus (ESB) approach is an approach that aims to integrate all systems of a company to one single system that can manage several different tasks. It is able to integrate heterogeneous systems as well. Rademakers and Dirksen [76] give an introduction into ESB's and claim that an ESB takes many of the advantages of other integration technologies and combines them. Figure 2.3 presents a schematic overview of the functionality of an ESB.

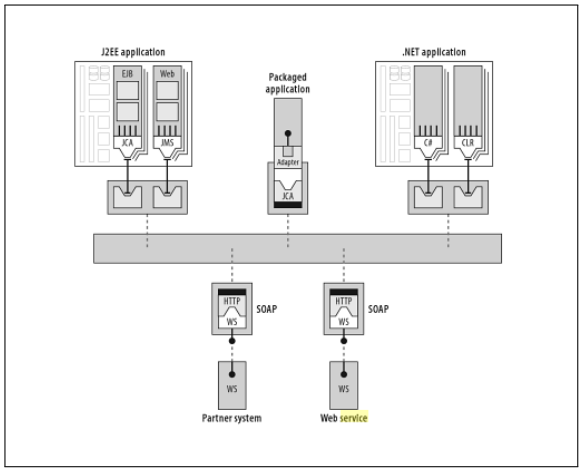


Figure 2.3: Schematic overview of an ESB from [76]

There can be several applications around the environment that are all connected to the ESB regardless of the communication type. With this premise, they can communicate with each other via the ESB acting as a heterogeneous interface.

Automation Service Bus

Another approach for implementing a heterogeneous environment is the Automation Service Bus (ASB) [19]. It is based on the idea of an ESB as described in the preceding paragraphs. The speciality of the ASB is its domain. The ASB is located in the discipline of heterogeneous engineering processes - in Automation Systems Engineering to be precise. It incorporates the systems that are involved in an Automation Systems environment. Chapter 3 gives a deeper introduction to the concept of ASB and introduces into the use case that is targeted by this thesis.

2.2 Performance Testing

This section introduces into Performance Testing. We see Performance Testing as a part of Performance Engineering and we present this interconnection. We further present an overview of the types of performance testing and introduce how to perform the techniques on projects.

First of all, we consider Software Performance Engineering and Software Performance Testing and see how it is defined. Woodside et al. [90] define Performance Engineering as follows:

> "Software Performance Engineering (SPE) represents the entire collection of Software Engineering activities and related analyses used throughout the software development cycle, which are directed to meeting performance requirements."

Another definition is from Williams and Smith [85]:

> "SPE is a comprehensive way of managing performance that includes principles for creating responsive software, performance patterns and anti-patterns for performance-oriented design, techniques for eliciting performance objectives, techniques for gathering the data needed for evaluation, and guidelines for the types of evaluation to be performed at each stage of the development process"

In this definitions we see that Software Performance Testing is included in SPE. We now consider Software Performance Testing in a detailed manner. In SPE we can differ between two basic approaches, measurement-based and model-based as Woodside et al. state in [90]. We now introduce these two.

2.2.1 Measurement-based Performance Testing

As of it's nature this type of performance testing can only be applied after the System Under Test (SUT) is already available. One can only run the tests if the application is ready to perform tests on it [1]. The SUT is started, tests are performed and tools based on this approach measure metrics during the execution. Afterwards, one can have a look on those metrics and can analyze the behavior of the SUT.

There is already some research on this type of Software Performance Testing approach. Arlitt et al. [4] performed a performance analysis on a Web-Based Shopping System. They emphasize the importance of retaining customers based on a research of Reichheld and Sasser [77] and state, the best way to keep business working is via quality. Beside business factors as stated from Lee et al. in [77] and [58] they state that also technical factors are important to serve a "pleasurable shopping experience" [77]. They imply that this can be reached by ensuring no long delays to make a purchase, ease and promptness in finding goods or services, etc.). To that end performance analysis is one important part of gaining and preserving quality of a system. They observed a Web-Based Shopping System that basically looked like figure 2.4 describes.

For that, Arlitt et al. investigated the data obtained by an E-Commerce site. They characterized the workload and applied clustering techniques to get knowledge about the user's behavior and its impact on the system performance [4].

[1] In sense of being able to get metrics of the system

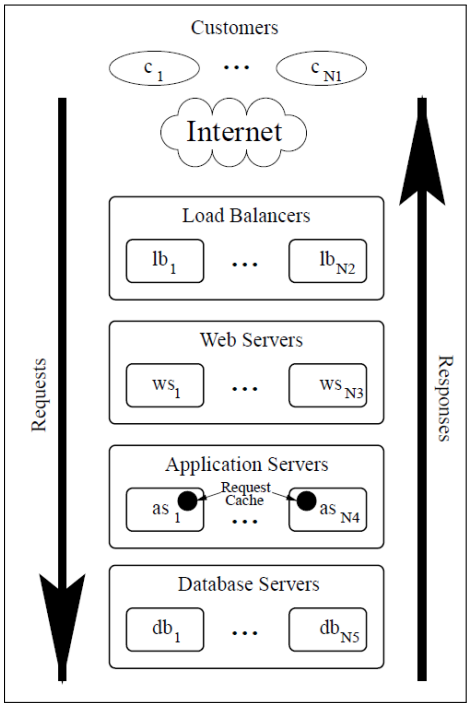


Figure 2.4: Multi Tier Tested System from [77]

Another study concerning Measurement-based Performance Testing is done by Avritzer et al. [6]. It also deals with workload characterization. They identify workload characteristics and apply their approach to a system that was enhanced to a large industrial system. They were able to identify a bottleneck that would have had "significant consequences" if it were undetected [6]. Avritzer et al. did even more research in the field of Performance Testing. Avritzer and Larson [7] introduced a technique called "Deterministic Markov State Testing" that tests an application and they further report the success of the new technique. Avritzer and Weyuker [9] extended the approach with three algorithms for test case generation and stated that basically every software system that can be modeled by a Markov chain can be tested. They present a study in which they apply their approach on five systems and show the validity of the approach. Furthermore, Avritzer and Weyuker [10] focused on the ability to compare two software systems. They present an approach which generates an application independent workload which is needed for the performance evaluation. With this approach one can easily compare two software systems and decide if the succeeding system is as performant as the original system. They substantiate their work with providing insights to their experiences they gained during the investigation. Avritzer and Weyuker [11] concentrate on a telecommunications project to figure out when to restore a system. In another research, Avritzer and Weyuker [12] combine the measurement-based approach with a model-based approach and state that this brings out the best of Performance Testing. Avritzer et al. [5] present algorithms for detecting cautious trends of metrics, such as response time. They trigger software rejuvenation before the metric provokes

a noticeable loss of performance, so that the user does not recognize any performance issues.

Barber [16] describes an approach to create load models if there is incomplete data. Barber [15] covers a broad range of Software Performance Testing topics, such as strategies, metrics or tuning.

2.2.2 Model-based Performance Testing

The second approach is called Model-based Performance testing. In this approach one applies methods to model the future system and to predict future performance. That implies that one can identify possible bottlenecks in advance and can prevent the engineers to make bottleneck-causing design failures.

Balsamo et al. [14] present a survey in which they investigated research in the area of Model-based Performance Testing. The work of Lazowska et al. [57] and Smith [79] revealed a need for earlier performance processing. In the survey Balsamo et al. name some "preliminary approaches", such as Hoeben [48], King and Pooley [56], Pooley [70] and Pooley and King [71]. They further categorize the methods for Model-based Performance Testing and name the following categories (categorization from [14]:

- Queuing Network-Based Methodologies (pioneered by Smith [80, 81]) [26, 27, 42, 44, 54, 62, 67, 68, 91]
- Process-Algebra-Based Approaches [13, 18, 40, 46, 47]
- Petri-Net-Based Approaches [17, 56]
- Methodologies Based on Simulation Methods [3, 29]
- A Methodology Based on Stochastic Processes [59]

Balsamo et al. then make a comparison and classification of the collected methodologies. The outcome of the classification can be seen in figure 5 of [14].

2.2.3 Performance Testing Process

There are many ways to perform performance analysis on software projects. As the process itself is a very important factor in performance analysis, there is already some research about that. Denaro et al. [31] identified a process that they claim to be good for early performance testing in software projects. They developed on this because they say that "the most critical performance faults are often injected very early, because of wrong architectural choices" [31]. It consists of four main steps that are:

1. Use Case Selection (Performance related)
2. Mapping of Use Cases to technology and deployment
3. Stub generation

4. Test Execution (consists of deployment, workload generation, data initialization and reporting metrics)

Denaro et al. performed a pilot analysis of their proposed process and it worked well. They identified an empirical analysis as future work to save the validity of their approach.

Another proposed approach for executing performance analysis is from Meier et al. [60]. Figure 2.5 gives an overview of their proposed approach. They identified the following activities that are needed in a Software Performance Testing process.

1. Identify Test Environment (including hardware, software and network)
2. Identify Performance Acceptance Criteria (goals and constraints)
3. Plan and Design Tests (key scenarios and test data)
4. Configure Test Environment
5. Implement Test Design
6. Execute the Tests
7. Analyze Results, Report and Retest

That are the Core Performance Testing Activities that Meier et al. identified and proposed in [60]. This approach is used throughout the complete guide [60] and is extended in the further work if necessary (for example for Iterative Performance Testing, Agile Performance Testing or CMMI [2] Performance Testing conformity).

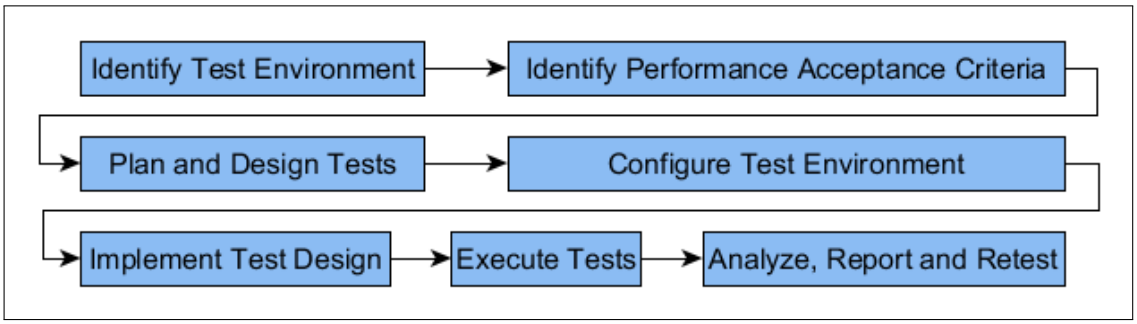


Figure 2.5: Performance Testing Process from [60]

Further, Barber [15] developed another process for Software Performance Testing which can be seen in figure 2.6. This approach already includes refining of tests and tuning of the system after an analysis of the results of the execution. They see the strategy more as aspects of Software Performance Engineering than as step by step tasks. So the mentioned aspects are (numbers relate to the numbers in the figure):

[2] Capability Maturity Model Integration [82]

1. Evaluate System (requirements, expectations)
2. Develop Test Assets (documents and plans)
3. Execute Baseline/Benchmark Tests
4. Analyze Results (expectations met, bottlenecks caused)
5. Execute Scheduled Tests
6. Identify Exploratory Tests
7. Tune System
8. Complete Engagement (documentation, historical reference)

As we can see the basic concepts and workflows are fairly the same in all three seen approaches. They differ in a few special cases and situations but basically they propose the same strategy. Nevertheless, none of these approaches has been applied to perform Performance Tests and Analysis on a Heterogeneous Distributed System like chapter 3 describes.

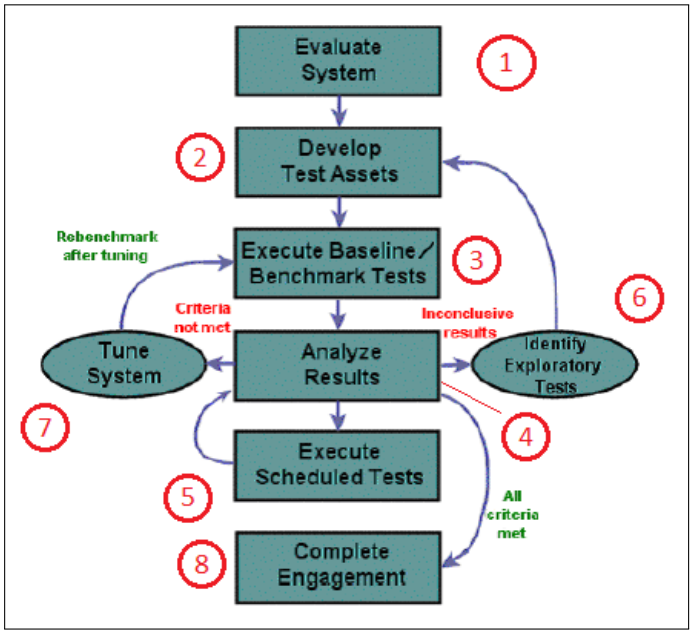


Figure 2.6: Performance Testing Process from [15]

2.2.4 Types of Performance Testing

In this section we introduce the basic and most important types of performance testing and their relating purpose. We give an overview when to use which type and what the properties and identities of each type are.

Load Testing

Load Testing is the task of exposing a system to a heavy load. The load can be controlled by either increase the number of users for example or by removing resources. Figure 2.7 represents a typical load created by a load test.

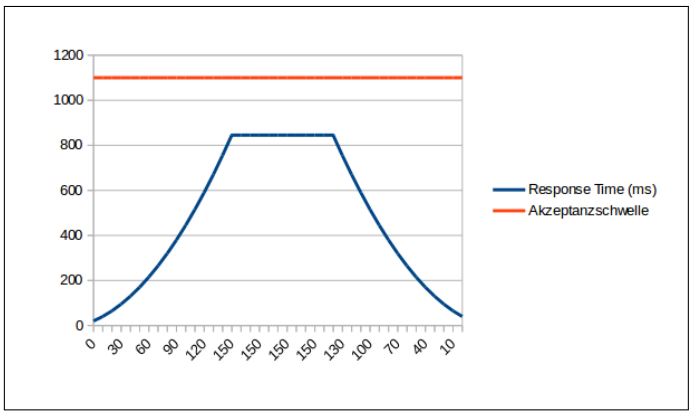


Figure 2.7: Schematic functionality of a Load Test

Load testing is also a big topic in research. Avritzer et al. deals with performance testing in his research. In [8, 9] they focus on generating test suites for load testing. Weyuker has further dealt with performance testing of component-based software [84]. Jiang et al. tried to automatically identify load tests problems. Menascé [61] also focused on load testing web applications. Draheim et al. [32] performed a load test on a web application by stochastic processes. Ghaith et al. [39] focused on anomaly detection of performance regression testing. Netto et al. [63] investigated the load generation in virtualized environments.

Stress Testing

Stress testing is a special kind of load testing. In Stress testing one tries to push the system on its limit and even over that limit. The system can get knocked out and therefore this approach is also known as negative testing. The target is to bring the system to fail and see how it recovers if the stress load is over and a normal load is applied. Figure 2.8 shows a typical stress test load.

On stress testing, there is not that much research explicitly because basically it is just a special case of load testing. Garousi [38] investigated stress testing of distributed real-time software.

2.3 Performance Testing Frameworks and Tools

This section introduces into the evaluation of frameworks and shows the base of the approach of tool evaluation of this thesis.

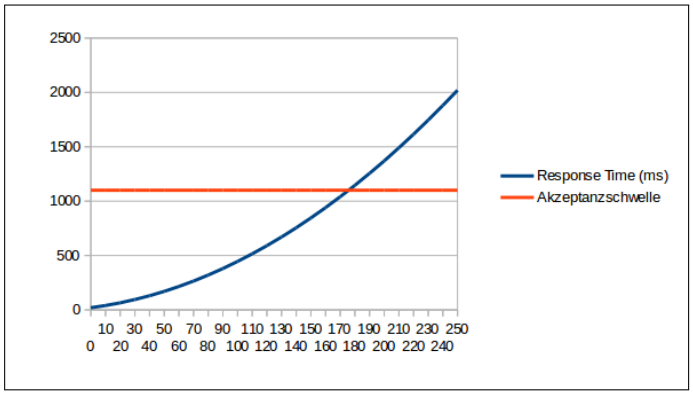


Figure 2.8: Schematic functionality of a Stress Test

2.3.1 Evaluation Process

It is important for choosing a tool to have a good strategy how to check the tools. If there is a procedure that is not comprehensible or can not be reproduced or smells like that the outcome of the tool evaluation will not be satisfying for the involved stake holders. So there is a need for a systematic evaluation process that is appropriate for a tool evaluation. Robert M. Poston and Michael P. Sexton took up that problem and developed an approach for evaluating and selecting testing tools in order to find the optimal tool for the aimed purpose [74]. They say that there is already a system like that, which is based on interconnections between tools and states that those interconnections are important during evaluation. This work ended up in IEEE Standard 1175 [1]. Robert M. Poston and Michael P. Sexton say that these approaches are successful because they minimize subjectivity and the work still accounts for tool-dependent factors for example. They also created forms because they say that a system can only be as good as the forms it provides.

In this section the approach of Robert M. Poston and Michael P. Sexton is described and explained.

2.3.2 General Process of Evaluation

The general process of evaluation consists of mainly 4 steps. They are namely [74]:

1. Identify and quantify user needs
2. Establishing tool-selection criteria
3. Finding available tools
4. Selecting tools and estimating return on investment

Figure 2.9 gives an overview of the main steps and the connections between each step to another step. The following paragraphs now explain each of the steps in detail and provide and overview over each step of the evaluation process.

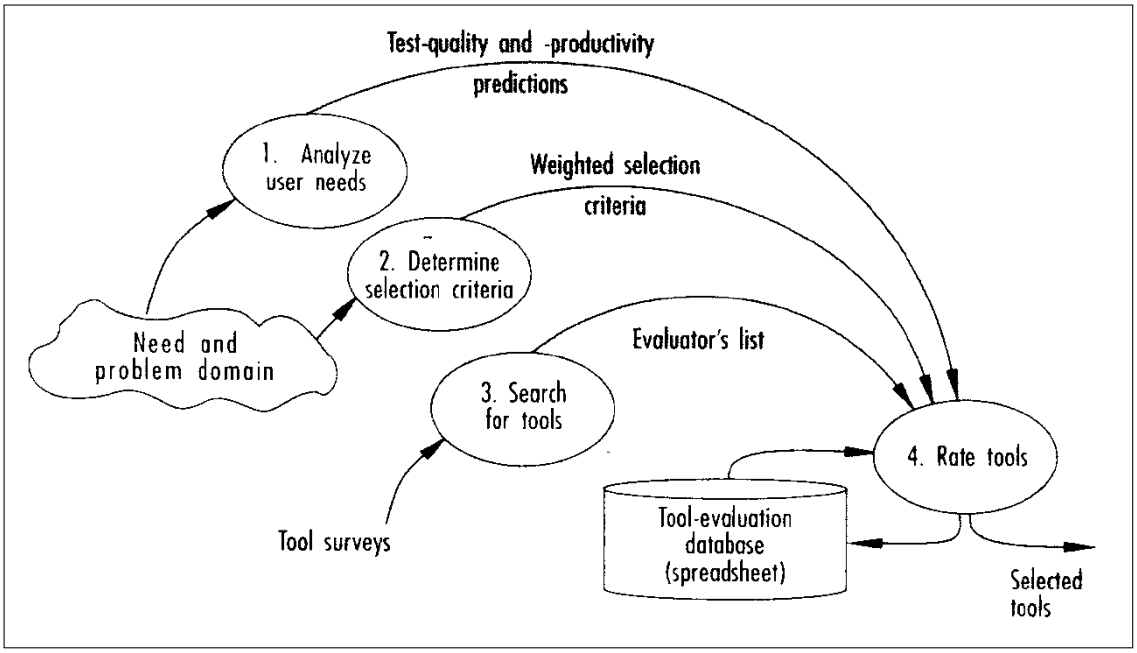


Figure 2.9: General Process of Evaluation of Poston and Sexton [74]

2.3.3 Identify and quantify user needs

First of all, you should verify with your managers if the need that they express is an actual need. This need analysis is important because it can prevent purchasing mistakes ([74] Page 3) The first step the authors describe is to collect statistic data from Quality Ensurance department (productivity and quality statistics). See [74] for a form for organizing the data collected by the needs analysis.

The second step is to determine the current testing effort in means of what is currently done in testing. First, see how much effort has been done in the most recent projects (measured in staff months) and use these values and a prediction for upcoming projects to estimate the amount required to ensure testing in a short term (of course assuming that the team will not use any new tools). Next step is to consider test quality and ask how many errors or issues the customer has revealed in the last release period. We will need this data to compare it later. Also get known about planned activities that could influence future testing activities and their results (e.g. reduce errors). Regarding the work of Robert M. Poston and Michael P. Sexton such activities could be for example: manual inspections or staff reorganizations. Same as before, this estimated test quality should show the future test quality if no other testing tools than up to now are introduced. The authors also mention also more sophisticated methods to predict future quality than simply performing an extrapolation. The mention two major publications for that: Function Points by Capers Jones [53] and test-quality measurement by Programming Environments [72] but for the sake of evaluating testing tools they say that those methods are too much effort for that purpose. They also make a statement if it makes sense to do a full tool evaluation at all. They therefore

consider a figure from the Business Week's special issue on quality [23].

They derive that if the predicted effort of the testing process is low or the failure-density values are lower than the company given threshold the evaluator should better go back and challenge the need of the full evaluation in the manager's need.

2.3.4 Establishing tool-selection criteria

Next part is to find criteria for the tool selection itself. Robert M. Poston and Michael P. Sexton therefore suggest a form. They group the criteria in four groups: general, environ-dependent, tool-dependent functional, tool-dependent unfunctional. The groups can be seen top down as every framework that could not pass the upper group, will not be evaluated any more. Further, the criteria are weighted to figure importance. We now take a closer look to the groups.

General Criteria

This criterion is the first filter for the framework selection and is basically compiled of two parts, the productivity gain and the quality gain as Robert M. Poston and Michael P. Sexton suggest. We fill in the expected values for each of the gains and add a weighting factor.

Environment-Dependent Data

This section contains the restrictions for the environment specific data. We select the maximum allowed costs for each of the items. If the company does not have any cost restrictions or you do not know this at all, we still can go on with the evaluation by fulfilling all other categories. This group contains costs for testing tools, organizational changes, platform changes and tool-interconnection changes.

Testing Tools The testing tools criterion represents the maximum amount of money the company will spend on testing tools or the company plans to spend (future budget). We only consider tools that can be afforded.

Organizational changes In this category the authors pick up items that can be found in the environment of the tool - in the organization of the company. They say that studies before 1984 state that normally implementation of the tools fail if the responsible persons do not consider organizational changes for the evaluation. [22, 30] In IEEE 1175bei these items are defined as Policies, Techniques, Work-product standards and Measurements.

Platform changes This section deals with any change of the underlying platform on which the tool should run on. It is possible to state that the tool has to run on the current platform and therefore one would fill zero into the criteria value field. Otherwise one adds the value the company accepts to pay for the new platform needed by each tool.

Tool-interconnection changes Tool-interconnection changes deals with the integration and communication of the tool with other tools.

Tool-Dependent functional criteria

This category deals with the tools themselves and their functional features. Robert M. Poston and Michael P. Sexton say that employees may meet for a brainstorming and hand the list over to the evaluators but this list may be incomplete or contain duplicated requirements and may not be standardized concerning definitions of vendors or industry. [74]

The evaluation clients can fill a form and hand it over to evaluators to give an overview of the features they expect the winning tool to have. The features that are voted the most is filled into the form of tool-selection. If there is a need for more than the given four features one can of course extend the list but the authors say that it is easier for the evaluation to keep this list short.

Tool-Dependent nonfunctional criteria

In this category the non functional criteria are listed. Performance is one of the non-functioncal criteria that might be interesting for the future tool. We further have to consider how to measure those criteria. Performance for instance can be measured in maximum response time that should be valid for all key features. One should define a reference use case or something similar to be able to perform an objective benchmark for measuring the performance.

As the authors say, more tricky factors are human factors. They further say that some of the human factors required by the evaluation clients are "red flags". [74] "User friendly" is one of those. Robert M. Poston and Michael P. Sexton also say that one should not concentrate on the very detail but concentrate on quantifiable factors (For example, time to lern/use a tool). Another nonfunctional criterion is reliability. One may measure the reliability in the tolerance of breakdowns in a given time slot.

In addition, to all mentioned factors one can also call some other factors in and extend the form. Such factors may be robustness or maintainability Robert M. Poston and Michael P. Sexton say. A report of the Software Engineering Institute [3] gives a detailed view on tool-dependent characteristics. [36]

Weighting

The next step is to add weights to the features. As the authors of [74] say there are two simple rules concerning weight assignment:

1. Every criterion must have a weight
2. No two criteria may have the same weight

The criteria are ordered with the most important feature having the highest weight and the least important feature having the lowest weight. The authors emphasize that this is "an easy but significant activity because it brings tool users to agreement on which requirements are most important. And it makes every potential tool user part of the evaluation" [74].

[3] http://www.sei.cmu.edu/

2.3.5 Finding available tools

At this point everything is well prepared to go into the search of available tools. The authors suggest that you make use of some publications that might help with creating a list of tools. They list for example the IEEE 1175 standard [1] as a base or a publication of Robert R. Poston [73], a reference guide of Jerry Durant [33] and a report of Dorothy Graham [41]. The authors suggest to use this lists to find a set of tools that might be interesting for the evaluation and the company. Further approaches are now presented.

Robert M. Poston and Michael P. Sexton suggest to use forms to express the organization in which the future tool should run. Another usage for the forms in the figures mentioned above is to create surveys out of them to get a knowledge for the evaluation (To see the form, see [74]).

As next step they suggest to contact the vendors, to get details for the products in the list. They say one should ask for a brochure, price list, filled tool forms, references. They argue that the kind of reply you may get will be a sign for the support the vendor is able or willing to give.

Further, one can use other evaluations as a base for a decision. Robert M. Poston and Michael P. Sexton say that one should be careful when using other evaluations because they might be biased for some reason. [74]

2.3.6 Selecting tools and estimating return on investment

After one has received the possible answers from the vendors one can start with rating the tools. The ratings should be percentage values and should represent the fulfilling value of the tool and feature. For instance, if tool A fulfills a requirement in 4/5 cases you should rate it with 0,8. The maximum that can be achieved is 1 and the minimum is 0 if the tool does net fulfill the requirement at all.

If one can not rate the feature exactly, a nestimation should be used instead. This estimation should represent the value of satisfaction with this feature with the tool. The values should then be multiplied with the weight and summed up to get a final score for the tool.

As last step, the authors suggest a meeting with managers and testers that are involved. The results should be discussed and everyone should feel free to contribute his or her opinion. They also state that if the people are involved in this process the implementation is more likely to succeed. [74]

2.3.7 Re-Evaluation

Once an evaluation has been finished, the winning tool will be implemented. There will most likely be a pilot project in which the new tool will be used the first time. In this project the tool's champions (those who voted for the tool at the final meeting) should participate. After the pilot project the project should be revisited and evaluated and so should the tool. The management will then get a feedback if the tool implementation performed as expected. [74]

2.3.8 Tool Studies

There are also other tool studies that are based on this approach. Illes et al. [51] define a list of criteria that can help when evaluating tools. Further, Kaur and Kumari [55] deal with a study of

two test tools to compare both and find the better tool. Ekaputra et al. [34] create an analysis framework for ontology querying tools and provide a decision helper with it. Pohjoisvirta [69] made a tool study for choosing a tool that supports test management.

2.4 Summary

There is much research on the integration of heterogeneous systems, such as for MoM [28,64] or SOA [50, 66] or Busses [76]. The ASB takes up the work of integrating heterogeneous systems in the Automation Systems discipline [19].

Further, there is already much research on the topic of Software Performance Engineering and Software Performance Testing. Many approaches deal with algorithmic parts [7, 9] and especially Avritzer et al. mostly deal with E-Commerce applications [5, 12]. Some work is also about telecommunication environments [7, 9–11]. The main items of the research gap we encountered are:

- There is no study about performance testing framework tool evaluation for our research environment for a tool selection
- No framework for performing performance analysis in the research environment has been designed or implemented
- There is no case study for applying a performance analysis framework in the research environment

We concentrate on the gaps of the recent research as described above. We provide a tool evaluation for performance tools with the approach of Robert M. Poston and Michael P. Sexton [74] and design a workflow and a candidate solution for Performance Testing in our research environment described in chapter 3. We further implement our designed approach and show that it is applicable to the environment.

CHAPTER 3

Research Context and Issues

This chapter introduces the motivating use case and expresses the research gap. Then the research questions handled by this thesis are explained and motivated.

3.1 Use Case - Change Management Workflow

In many systems many different disciplines are involved hat contribute to the holistic system in their own specialized area. These experts have favorite tools or tool sets and are familiar with them as they have already worked with them before and so they do not want to change the tools easily. These tools are mostly not compatible to tools of other disciplines and other experts' areas. That implies that the experts have no standardized way to communicate via the tools with each of the other disciplines. These heterogeneous environments are a big problem in terms of integrating different disciplines' tools and to provide a single fine waved working system.

The ASB as shortly introduced in section 2.1.1 seizes this problem and provides an approach to integrate heterogeneous tools in one single environment. Figure 3.1 presents the basic concept of the ASB as introduced from Biffl et al. [19]. We can see the different disciplines integrated via the ASB are Project Management, Systems Engineering, Software Engineering, Team Communication and an ASB Support. All these disciplines communicate via the ASB.

The ASB provides technical integration of tools via connectors as well as semantic integration via the Engineering Knowledge Base (EKB) and Engineering Database (EDB). The technical integration is accomplished by implementing connectors for each tool. They are now able to communicate technically with the ASB. Anyway this is only a part of the solution to integrate those hetereogeneous tools. The second important part is the semantic support for the tools so that the tools - respectively the engineers - of different disciplines are able to communicate with each other. Therefore, data is stored together with the models in the EKB/EDB and they do the mapping and semantic transformation.

The resulting advantage is that each of the tools, independent of the discipline it is originated, can communicate to all other disciplines. The experts can now focus on their own tools and can be sure other experts are able to interact with their work seamlessly.

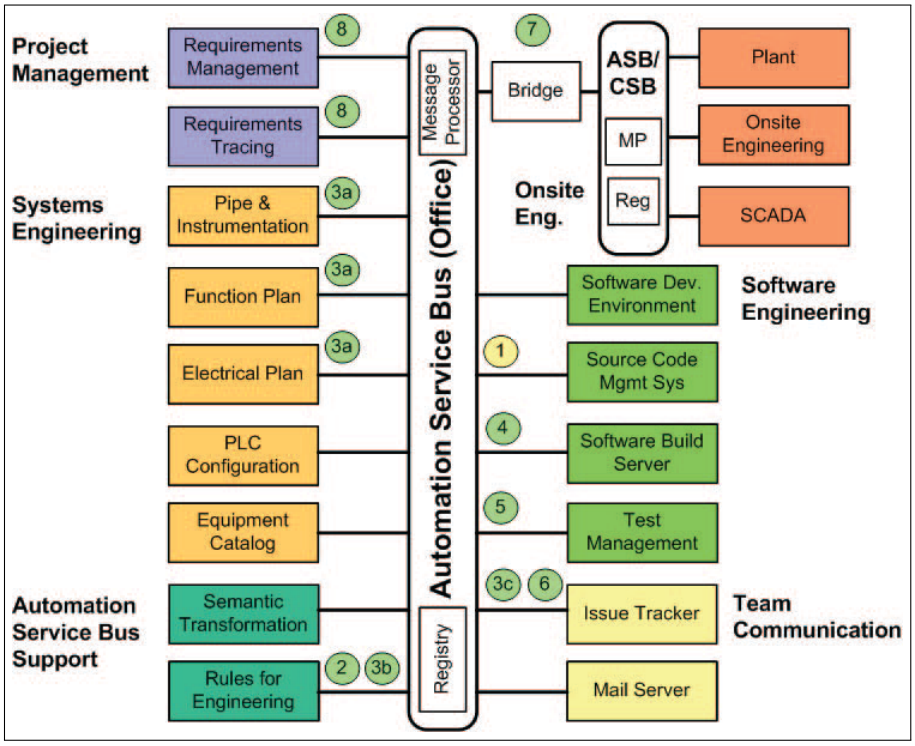


Figure 3.1: Basic Concept of the functionality of the ASB from [19]

Based on the ASB approach, one of the tools that interact with the ASB is a web application that provides some basic functionality to the users. It offers the possibility for project management, checkin/checkout of signals, user management and many other features. With the checkin feature for example the user is able to upload signal files and to check in those files into the ASB environment. This process is a time consuming operation and does not work in proper time with larger files. We present the web front end later on.

The motivating use case of this thesis is the change management workflow which is a complex and critical process in the Change Management Process. Figure 3.2 illustrates the change management workflow and the steps of it. The first step of the workflow is to start the checkin depicted with E1. E2 decides if there are changes in the signals at all. If the decision states that the signals are similar than the workflow proceeds with no change (E3). If changes are recognized the accept changes decision is executed. If the changes are not accepted no change (E4) is executed otherwise the signal gets either updated (E5), is recognized as a new signal (E6) or is deleted (E7). After this, a notification is sent. The workflow can then proceed with the next signal if any, as stated in E9. E10 represents the end of the workflow.

As stated in the preceding paragraphs, performance is an important aspect in the web application as well on the ASB itself. There are performance requirements for both parts and there is a lack of proving that these requirements are met. The target of this thesis is to seize this gap and to find a possibility to verify the performance of the web application.

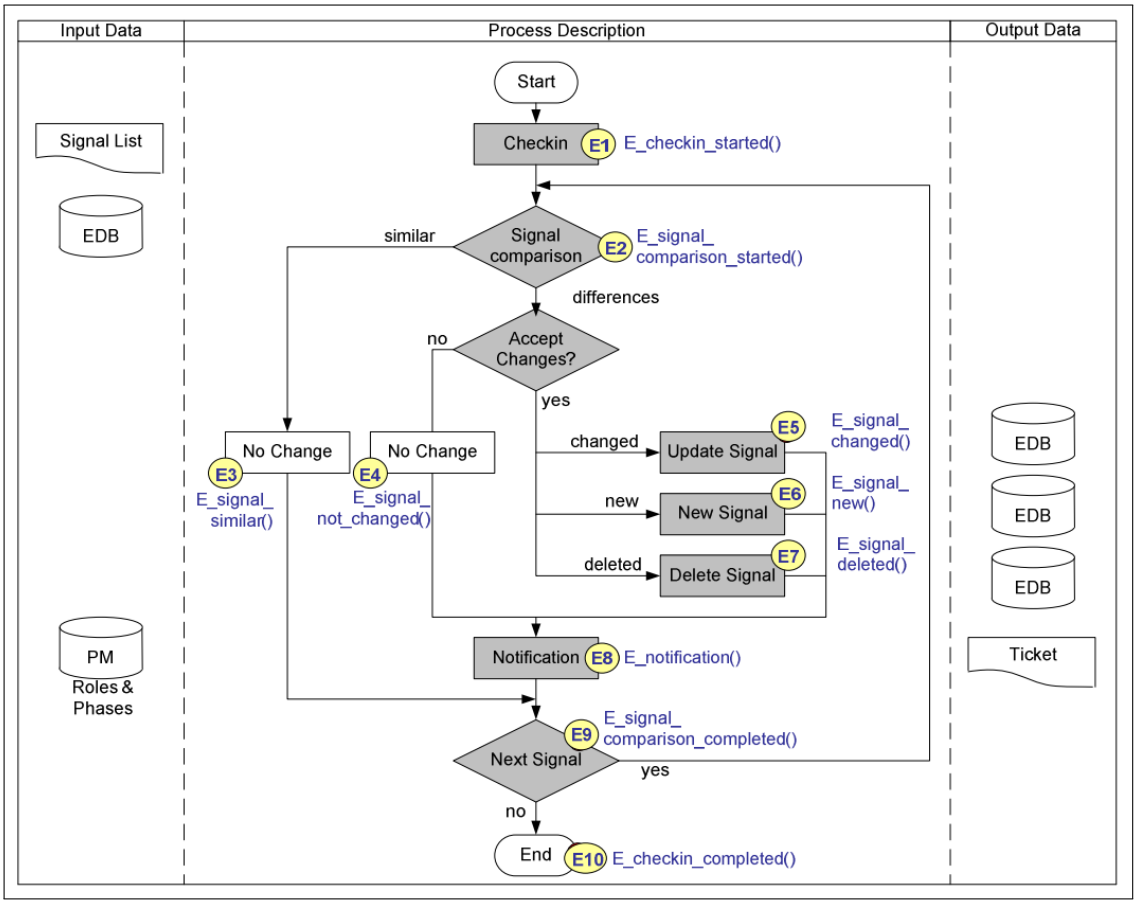


Figure 3.2: Change Management Workflow from [89]

We now outline the research questions (RQ) addressed by this thesis and explain the goals of this thesis. The research is driven by the use case described above. Therefore, the research questions are tailored to this research environment. The high level aim of this thesis is to investigate performance testing in a new research area - the area of heterogeneous distributed systems such as Automation Systems. The thesis' aim is to find a way to perform performance analysis in this environment in a sufficient manner and to find a solution for the existing gap. In the holistic picture of the thesis we identified three main research questions:

- How can we find a tool that enables us to perform performance testing?
- How should a process look like that guides performance testing in the environment?
- Is the proposed approach feasible and applicable?

These are the main research questions of this thesis. We now present and explain each question in detail.

3.2 Research Question 1 - Tool Selection

In chapter 2 we describe recent research. We can see that there are several approaches to perform performance analysis in existing work. They all deal with performance testing in different contexts and have different aims. There are further studies that deal with tool evaluation. The approach by Robert M. Poston and Michael P. Sexton [74] for instance presents an generic approach for evaluating tools. They present a strategic way to gain information of tools and select one of them. Based on this approach, many studies accomplish a tool selection. There are, for instance, studies that deal with test framework evaluation or simply choosing the better tool out of two. We count the following studies to this:

- Illes et al. [51], defining a list of criteria for a tool evaluation of software testing tools
- Kaur and Kumari [55] comparing two test tools to find the more suitable one
- Ekaputra et al. [34] creating an analysis framework for ontology querying tools
- Pohjoisvirta [69] finding a tool that supports best for test management

These examples of recent tool studies also show the context and the research environment they are performed in. This thesis is located in the context described above and in chapter 2, the environment of heterogeneous, distributed engineering environments. We describe this context in chapter 2. We can find recent studies in this research environment about applying methods of Software Engineering to this environment. There are several approaches that already deal with different processes and methods of Software Engineering and the application in the new environment such as Hametner et al. [45] who adopted test driven development process to industrial automation engineering. Further, Winkler et al. have several studies in this area such as [87] (deals also with adopting a test first development to improve automation systems engineering processes) or [88] (dealing with efficient unit testing). Winkler et al. [86] (creating a framework for automated testing of automation systems) is especially interesting for this thesis as it deals with the creation of a testing framework in the research environment. We could see that there is many research for performance testing as well as for dealing with Software Engineering processes and methods in the new research environment. The main goal of this thesis is to perform an performance analysis in the research environment. None of the studies we could find, deals with this topic. To be able to perform this analysis, we need a tool that supports that. A search on the market for available tools that offer performance analysis results in a very long list of tools which have various features and several targets. To make a good decision which tool to choose, we have to select a tool in an appropriate way. We ensure the appropriateness by using approved methods such as Robert M. Poston and Michael P. Sexton [74]. This leads us to the fact that we have to be sure about the criteria a potential tool has to fulfill in order to be suitable for us. This need reflects our first research issue:

RI 1.1: *What are the requirements for performance testing in the research environment?* The goal of this RI is a list of requirements that reflect the user needs to be able to select a must

suitable tool and this is the contribution of this RI.

The answering of research issue 1.1 brings a list of requirements that the tool has to fulfill. Once we have received that list, we are now aware of what our tool should be able to do and what features it should have. With the already achieved list of available tools we are now able to select a tool. The selection itself is split up into two parts. The full list of tools is reduced to a short list by checking mandatory features. The short list is then evaluated in detail to get the most suitable tool. This process reduces the long list by using the criteria of the previous determined list and performing the tool evaluation. The execution of the evaluation maps each of the tools and each of the criteria to a value that reflects the compliance of the tool to the criterion. This leads to the next research issue:

RI 1.2: *How do the available tools fulfill the requested requirements?* An executed evaluation and a table that shows the fulfillment of the tools regarding the requirements is expected as contribution.

By answering this research issue, we get a table of values with the ratings of the tool relating to each criterion. As we want to perform a performance analysis, we are not done so far, but we need an overall appraisal of each tool which brings us to the last research issue of the first part of the thesis:

RI 1.3: *Which of the available tools or set of tools is the best for the environment?* One of the tools comes out of the evaluation with the most point. This is the most suitable tool and is expected to be the contribution of this RI.

3.3 Research Question 2 - Test Framework Design and Implementation

In the previous section we deal with the evaluation of a performance testing tool in order to be able to do performance analysis on this field. To that extent we achieved a most suitable tool and with this tool basically have the technical abilities to do performance analysis. In chapter 2 we show processes that concentrate on the execution of performance analysis and the relating accessary parts. We present for instance the approach of Denaro et al. [31] which deals with early performance testing in projects. Another proposed approach of implementing performance analysis is from Meier et al. [60]. It also deals with the process of performance anaylsis and extends the basic approach step by step. Barber et al. [15] present an approach that already contains test refinement in their work. These approaches are general and do not specialize on a certain type of projects or environments. A more specific approach is by Winkler et al. [86] For our purpose of executing performance analysis in the research environment, we also need a process or framework that leads and controls the overall procedure and offers an appropriate way to do so. This results in another research issue:

RI 2.1: *How does a framework for performing performance tests look like with the most promising tool and in the research environment?* The goal of this research issue is to design a framework that is able to support and guide performance analysis in the research environment.

After the design of the framework is finished, there exists a method for performance analysis in the specific environment of the research area. Based on that approach, we claim that it is possible to implement the framework and execute a reasonable performance analysis on a system unter test of the research environment. As stated here, the framework has to be implemented in order to be able to be executed. To that end, we also need to define test cases that have to implemented and tested. All that leads us to the next research issue:

RI 2.2 *How can the designed framework be implemented in the research environment?* The outcome of this research issue is the implementation of the designed framework. As a contribution this framework implementation is performed with the most suitable tool.

3.4 Research Question 3 - Test Framework Evaluation

We have presented so far a tool selection process that selects a most suitable tool out of a long list of available tools by applying criteria and evaluating the score of each tool. Further, we created a framework based on recent approaches that is able to lead and control performance analysis in our research environment. We can apply those approaches to perform the analysis on our system under test and we get results in terms of performance testing. The only thing which is still open, is to show that the presented approaches are feasible and work for our environment as intended. For that reason the last research issue comes up:

RI 3.1 *Is the implemented approach feasible and does it work properly in the research environment?* The contribution of this research issue is on the one hand a feasibility study to show the framework is working properly and on the other hand a set of results and improvement suggestions for the system under test.

CHAPTER 4

Solution Approach

This chapter presents our solution approach concerning the research questions stated in chapter 3. In each section we propose a solution approach relating to the three presented main research questions. The sections themselves introduce approaches for the fine grained research issues of each question. We start with the approach for the tool evaluation, continue with the design of the test framework and its approach for implementation. This chapter closes with the proposed method for the case study which shows the feasibility of the described framework.

4.1 Tool Selection

This section describes the evaluation method for the tools that are currently available for performance testing that is based on the approach of Robert M. Poston and Michael P. Sexton [74]. In their work they describe that at first there is a need to find out which tools basically concern the study. We have to find out which tools are available on the market and which ones do not fit at one view and so create a list of tools of concern. The other tools can be ignored because they will not be suitable at all and will just unnecessarily expand the evaluation because they will be discarded in the evaluation anyway. Robert M. Poston and Michael P. Sexton furthermore describe that the criteria taken for the evaluation should come from the future users or the environment in which the tool should be used. The output hereby is a list of requirements that clearly state the requirements by the users and an additional weight to emphasize the importance of each requirement. We now reconsider the phases of the approach of Robert M. Poston and Michael P. Sexton [74].

4.1.1 Phases of the Evaluation Process

We base our evaluation process on the earlier mentioned work of Robert M. Poston and Michael P. Sexton [74]. In this work they describe four main steps for a good tool evaluation study. Figure 4.1 shows the steps.

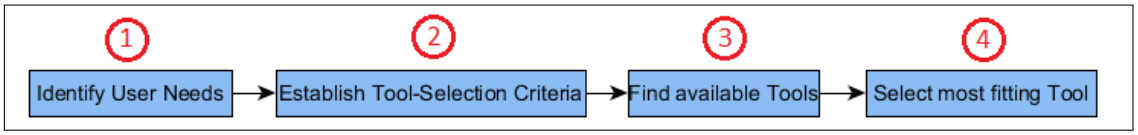


Figure 4.1: Main Steps of the Approach of Poston and Sexton [74]

The first phase is - as stated in the above paragraphs and highlighted with number 1 in figure 4.1 - the phase to find user needs. The second phase serves to create criteria out of the collected user needs and requirements stated as number 2. The output is a full featured list of needs that the winner tool has to support. This list consists of mandatory requirements and optional requirements. The third phase - marked with number 3 - is to find all available tools and list them up for evaluation. After the evaluation, we choose the tool with the best rating - number 4 in figure 4.1. This step consists of a pre-selection of tools based on the mandatory requirements to get a short list. This short list is then processed in order to find the most fitting tool. We now explain how we rate the tools.

4.1.2 Weight Strategy and Rating

For the purpose of emphasizing one criterion's importance, we have to create a possibility in our rating process. Therefore, we introduce weights which have to be assigned to each of the criteria. The weighting system is related to the Rank-order weighting method of [83]. To increase the relative importance of the more important weights, we increase the difference between two consecutive weights by 1. Therefore, we define four weight types as table 4.1 shows.

Weight Name	Weight Value
Critical	7
High	4
Medium	2
Low	1

Table 4.1: Overview of all possible weights in our approach

Further, we have so called "must have" requirements. The "must have" requirements have to be fulfilled by a tool in order to be further evaluated. If a tool does not fulfill a single "must have" criterion it can be immediately discarded. The other four weight categories we define as follows:

- **Critical:** We define the highest possible weight with the terminus of critical. We assign the critical weight the value of **7** to show that this criterion is part of the most important category.
- **High:** Second important weight is high with the mapped value of **4**. As the name suggests, this represents a high category.

- **Medium:** This weight category shows a medium importance is therefore assigned with a value of **2**.
- **Low:** Low weight states that there is not very much importance on that criterion but there is at least importance. We map this to a value of **1**.

After we have now introduced the weights for the rating process, we can continue with the process itself. The rating process calculates a final value for each tool depending on the grades given from the evaluators for each criterion and tool. Figure 4.2 shows an example for the rating process.

	Weighting Factor	Tool 1	
		Rate	Score
Group 1 – Category 1			
1.1 Criterion A	1	0,5	0,5
1.2 Criterion B	2	1	2
Group 2 – Category 2			
2.1 Criterion C	4	0,5	2
2.2 Criterion D	7	0	0
		Total:	**4,5**

Figure 4.2: Rating Calculation Example according to [74]

At first the weightings are filled in into the "Weighting Factor" column according to the collected weights of the future users. MH stands for the must have features. The others are filled with the mapped values to each weight. After the evaluation of a tool is finished the rate column can be filled. It represents a value between 0% meaning the tool does not fulfill this criterion at all and 100% saying this criterion is completely fulfilled. The score of each single criterion can now be calculated with the formula $score = weight * rate$. The scores are then added to a total score as seen in figure 4.2. This represents the total score of the tool according to the chosen criteria and weights. The higher the score is, the better is the tool. The total score for tool 1 in the example would be $9,35$. We further see in this example how tools are handled that do not fulfill at least one of the MH criteria. The do not get graded at all and immediately fall out of the evaluation which ends up with a score of 0. We now introduce the criteria categories and the criteria itself which are taken into account for this study.

In advance, the criteria which are not rated as must haves are rated in one of the two following ways. The first one is just to assign a value between 0 and 1 (comma values are valid with a maximum of 2 decimal places) regarding to the fulfillment level of the criterion. The second way is to rate according to Fay et al. [35]:

- 1.0 if a criterion is fully fulfilled (or a question can be answered with "yes")
- 0.5 if a criterion is partly fulfilled (or a question can be partly answered with "yes")
- 0.0 if a criterion is not fulfilled (or a question has to be answered with "no")

The values generated from these two approaches are further used for the calculation mentioned above to calculate the score of a tools' criterion fulfillment. The tool (or set of tools) which the highest total score is the most promising tool and thus is used for the further thesis.

4.2 Test Framework Design and Implementation

The last section showed the process of approaching a most promising tool for the research environment, how the criteria are gathered and how they are applied. In this section, we show how the most promising tool is designed to be applied on the described use case in chapter 3 and develop a test framework for this purpose. We further present the tests that are used to implement a prototyping performance analysis for the check in use case.

4.2.1 Test Framework

According to Winkler et al. [86] we describe a test framework for our purpose. This framework helps to perform the performance analysis on our research environment and provides the basic workflow from the need of testing to the test results. Figure 4.3 shows the graphical overview of the test framework.

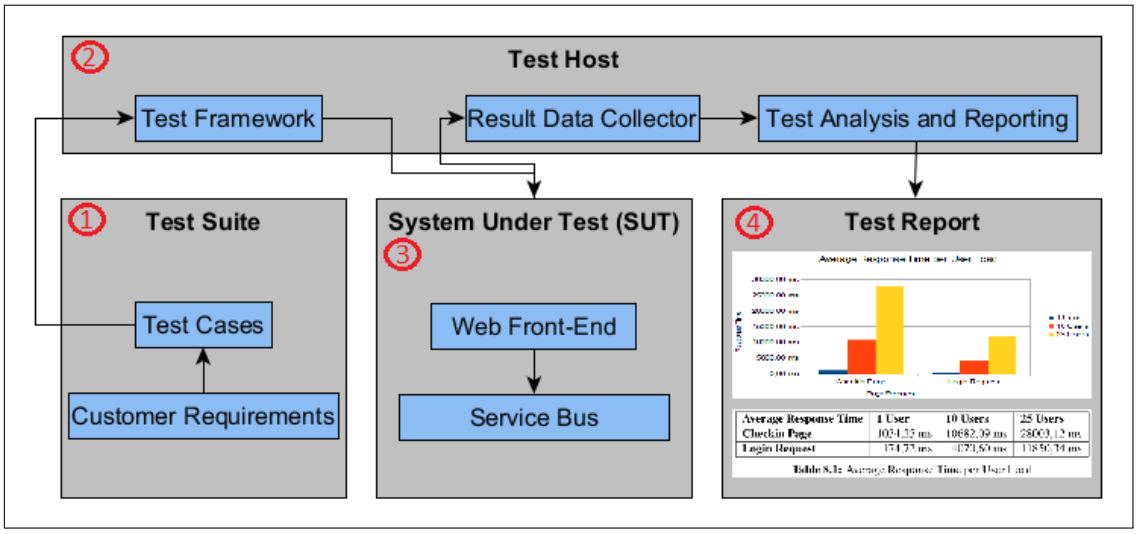


Figure 4.3: Test Framework Design according to [86]

The components of the framework are now explained. The **Test Suite** defines and describes the test cases according to the customer requirements (number 1). Then, the test cases are passed to the **Test Host** (number 2) where the test framework is located. The tests are then executed on the **System Under Test (SUT)** which consists of the web front-end and the underlying service bus (number 3). The data that is generated during the tests is collected and saved by the result data collector and further, handed over to the test analysis and reporting component. Based on

this, the **Test Report** (number 4) is created. We now continue with the use case of this thesis relating to the test framework.

4.2.2 Use Case

The use case we use for the performance analysis is the change management use case respective the checkin process of the web application front end. This use case is described in chapter 3. This use case deals with the change management mainly for signals that are treated by the application. The checkin workflow consists of three main steps:

1. (File) Upload and Transformer Configuration
2. Merge Signals
3. Review
4. Update Signals in the System

These three main steps also shown in figure 4.4:

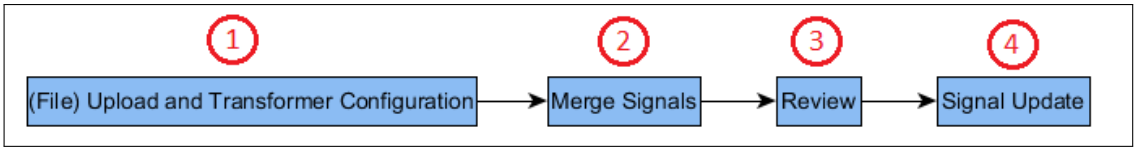


Figure 4.4: Visualisation of the three main steps of the Checkin Workflow

In the first step a signal file must be uploaded in order to be allowed to proceed. Figure 4.5 shows the first step. Further, the user has to choose the type of the signal file to upload (number 1). The possible options are currently the types of plc, elp or cul[1]. According to the file size the upload can take a while. Once the upload is finished, a so called path has to be chosen in a drop down box. The default value is plc. The next action is to proceed to the next step by clicking the "Next >" button.

The second step of the checkin process is the merge signals view (number 2). Figure 4.6 shows the first step. In this view the user can see an overview of the uploaded file and check if the file data has been read correctly. Further, in this view the user is able to set strategies for signal merging such as adding new or keeping old signals or replace signals. Again the user can proceed by clicking the "Next >" button.

In step three the checkin process displays the uploaded contents and the results of the parsing (number 3). Figure 4.7 shows the first step. The selected transformer as well as the selected path are displayed and statistics about the signals that were uploaded in the file. The user can review the values and check if everything went ok during the checkin process. Step four is processed after finishing the wizard (number 4).

[1]These are types of signals that can come from the heterogeneous environment in this use case.

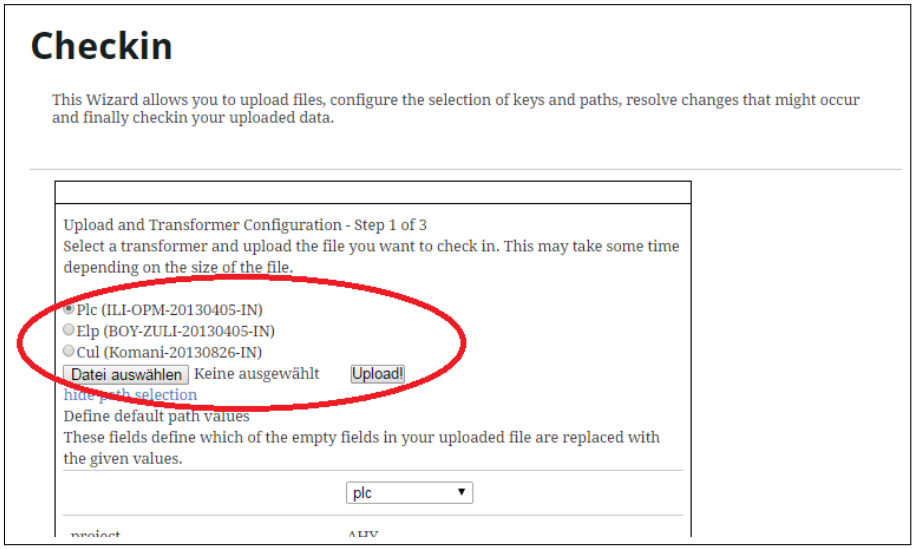


Figure 4.5: Screenshot of the first step of the Checkin Workflow

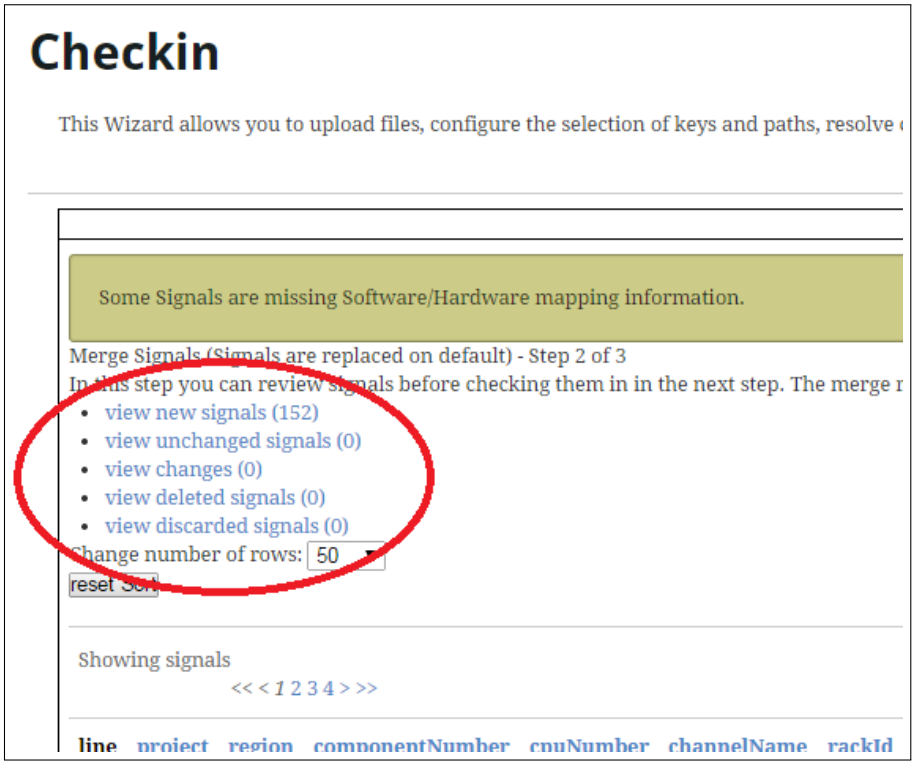


Figure 4.6: Screenshot of the second step of the Checkin Workflow

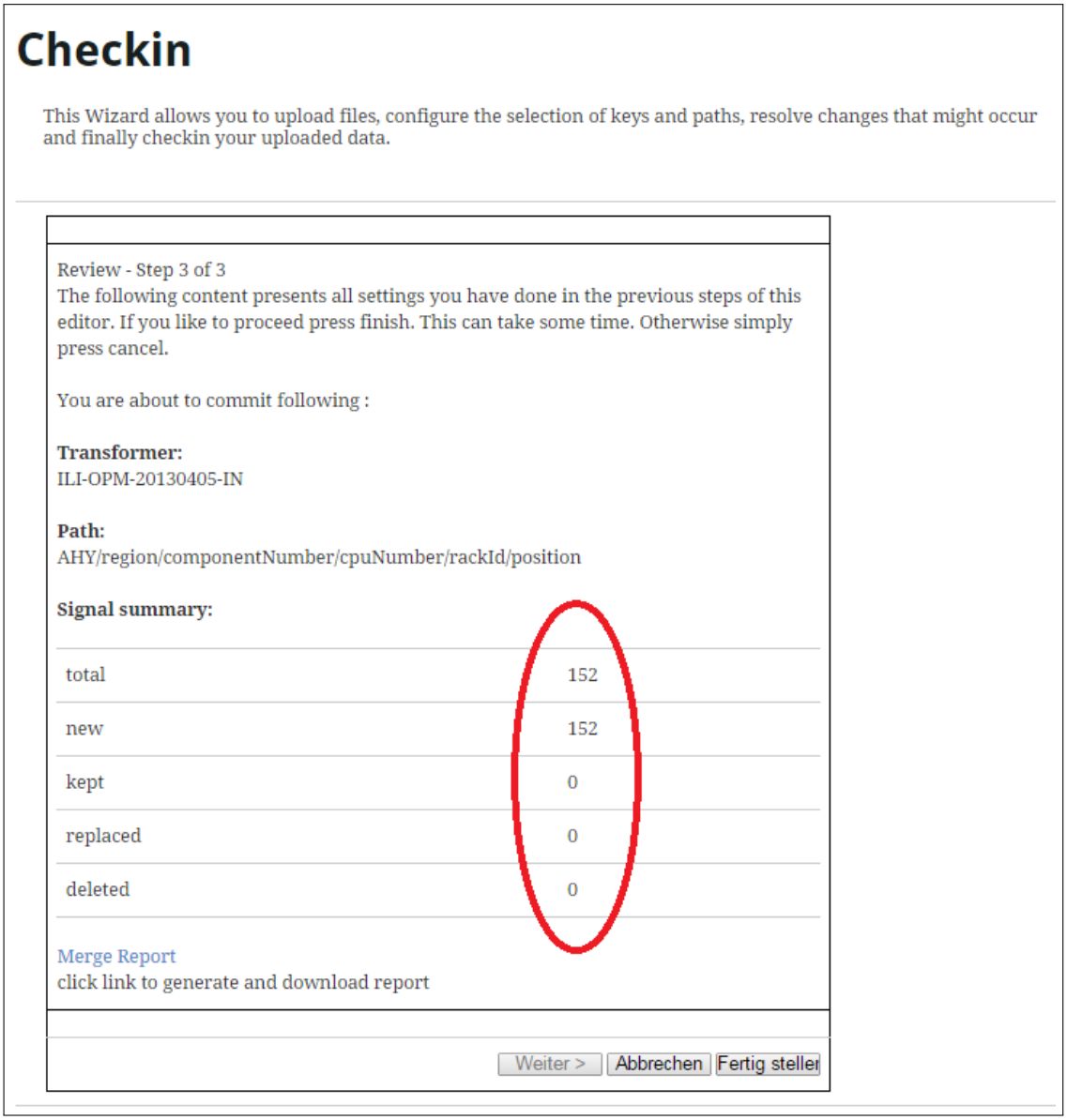


Figure 4.7: Screenshot of the third step of the Checkin Workflow

For the testing purposes, we can see some potential pitfalls for performance. In the first step the user has to upload a file. This may impair the overall performance of the application since the files may be large on the one hand and on the other hand the server might be overloaded with a large number of parallel file uploads. Also the parsing of the file to gather structured data may be time consuming and is therefore a possible pitfall for the application.

4.2.3 Planned Tests

We have now described the use case and its steps. We showed the main steps of the process of checking in signal files and provided ideas for possible pitfalls for the performance tests. For the tests we aim to approach a full load test step by step. We make use of the described approaches of section 2.2. We test the change management use case as described in chapter 3. This use case has been identified with the industy partneres as most critical process and is representative for the industry settings. Therefore, we aim five steps for testing. They are:

1. **Type 1** - Perform the Checkin Process only until Step 1 is displayed (figure 4.5)
    a) Request Page before critical steps with one user for 30 minutes
    b) Request Page before critical steps with 10 users for 30 minutes
    c) Request Page before critical steps with 25 users for 30 minutes
2. **Type 2** - Perform the Checkin Process completely as stated in figure 4.4 until the wizard is finished (figure 4.7)
    a) Perform Checkin Process with one user for 30 minutes
    b) Perform Checkin Process with 10 users for 30 minutes
    c) Perform Checkin Process with 25 users for 30 minutes the service breaks
3. **Type 3** - Perform the Checkin Process completely as stated in figure 4.4 until the wizard is finished (figure 4.7)
    a) Perform Checkin Process with a steadily increasing amount of users until the service breaks

These tests are especially crucial for performance testing as they can be performed often in a row and also in parallel which can cause the performance to decrease. We use the recording feature of the most promising tool (if available) to record the test cases via the browser. Therefore, we set up a proxy server within the tool and record all requests that are performed during each tests. That offers the possibility to get all requests that are made indirectly, such as AJAX requests, as well.

To achieve a good collection of data to interpret the results, we measure the following metrics depicted in table 4.2 in each test:

The testing procedure will consist of seven tasks. These tasks are shown in figure 4.8.

Metric	Measurement	Requirements
Response Time	Full load time of page	< 5000ms
User Count	Number of active threads	1/10/30, depending on testcase
CPU Usage	% of CPU-Load	< 80%
Memory Usage	% of Memory-Usage	< Max Memory
Max Memory	Memory provided by JVM	< 1,5GB

Table 4.2: Measured Metrics

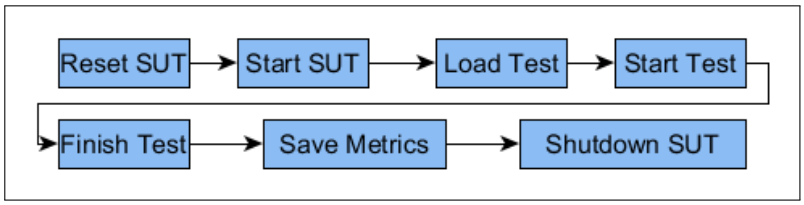


Figure 4.8: Test Procedure

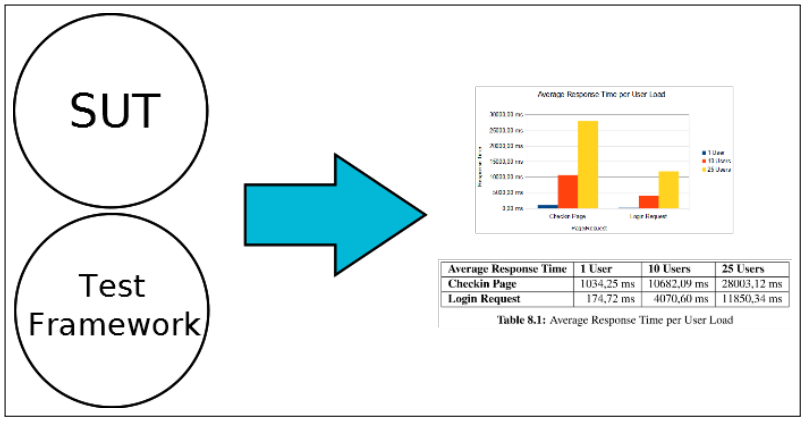
First, the SUT[2] has to be reset so that as a second step it can be started again. The test is then loaded and subsequently started. After the test has finished, we save the metrics the tool collected and shutdown the SUT.

The framework of this thesis designs those tests with the most promising tool and implement them with it. In the next part we present the results of the framework performance.

4.3 Test Framework Evaluation

The last part of this thesis is a case study performed on the web application prototype of the CDL-Laboratory. This application is used as the SUT and the evaluation refers to this application and to be precise on the use case described in chapter 3. We start with the implemented tests which are described in section 4.2 and execute all proposed tests in the research environment on the SUT. The gathered data is then collected and evaluated. We compare the data and provide aggregated values as well as expressive charts.

[2]System under Test



Average Response Time	1 User	10 Users	25 Users
Checkin Page	1034,25 ms	10682,09 ms	28003,12 ms
Login Request	174,72 ms	4070,60 ms	11850,34 ms

Table 8.1: Average Response Time per User Load

Figure 4.9: Case Study Workflow

Figure 4.9 illustrates the workflow that we used in the case study. The SUT and the test framework is used, the tests are executed on the SUT and generate data which is transformed into charts and data tables.

CHAPTER 5

Tool Evaluation Results

This chapter reports the results that we determined in the first part of this thesis - in the tool evaluation. It gives further answers to the research question 1 and its containing research issues. It lists the results of the evaluation and argues about them, according to the announced solution approach of chapter 4. The outcome of this chapter is the answer to the research question 1 and the execution results of the evaluation procedure which presents a most promising tool (or set of tools) for the further study.

5.1 Application Requirements

The first phase of the evaluation study is to find the user needs. This section answers the research issue 1.1 stated in section 3.2. The evaluation criteria are the result of a discussion of the respective experts of the CDL Lab. The discussion led to several user needs. We collected the user needs and selected the most appreciable criteria according to the experts. Therefore, in our environment we identified the following requirements.

5.1.1 Criteria Categories and Criteria

For clarity reasons, we introduce categories to which we classify each of the concrete criteria. Each of the categories holds a set of criteria that may further be split up into sub-criteria (such as in “Measureable Metrics”). Each criterion is tagged with a weight for the calculation and further on with a rating which together leads to a score. In the following we describe the categories and the contained criteria. The criteria categories are:

- **General:** This category contains general criteria which are important to the tool evaluation.
- **Performance (Testing) Issues:** This category holds all criteria that are related to the performance testing itself.

- **Interoperability:** All criteria that are listed in this category are important in concerns of interoperability with other tools.
- **Usability:** The last category contains usability aspects which are important in terms of using the tool.

Before we go into detail with each category and the relating criteria we present the must have criteria for the first selection step.

5.1.2 Must Have Criteria

In this paragraph Blocker Criteria are explained. That means if a tool does not fulfill all these requirements, it is immediately sorted out and is not be taken into account for further analysis. We now present the must have criteria.

Evaluation Availability

This criterion checks whether there is a version of the tool available for downloading and for our evaluation and testing purpose. If there is a version available for download, the tool is ok for this criterion and if there is no version available the tool fails.

Platform Availability

We expect the tool to run on different platforms. The important part expressed throughout this criterion, represents that we expect the tool to run at least under Windows [1] or current available Linux Distributions [2]. If the tool does not support either Windows or Linux the tool fails this criterion.

No exclusive cloud usage

We test the tools and their description if there is the possibility to run the tools from a private computer. We do not allow services that are only accessible through a cloud as we want to run the tools locally as well. If a tool is only available through a cloud and can only be used in this cloud the tool is rejected and fails this criterion.

Maintenance and development activity

At this criterion we check if a tool is still maintained and has an active development or bug fixing. This is important to us as we need a tool that is working without any critical errors and if a critical error raises up we need it to be fixed. Further, also open source tools that are not actively maintained or do not ensure bugs to be fixed are rejected because we do not want to invest any amount of work to fix eventually raising bugs on our own. We measure that with the download rate and release cycles. If a tool can not show an active maintenance and bug fixing it is rejected and fails this criterion.

[1] Windows 7
[2] Ubuntu 14.04.1 LTS

Environment Compatibility

We need to test if a tool is compatible with the current research environment. We therefore check if the tool is runnable under the configuration that is state of the art in the research environment. This requires to check if the tool is runnable with Java Development Kit 7 (JDK7) and can use modern browsers for web testing (Firefox 33) [3]. If a tool does not comply these requirements it is rejected and fails this criterion.

Data Export Possibility

As we want to use the data gained by the tool (either in the tool itself or with external tools) there is a need for exporting the data. This criterion checks whether a tool is able to export the data for further usage in any way. If a tool provides a way to export the data for further usage it passes this criterion and if it does not it fails and is rejected.

Time to productivity

This criterion states that a tool has to be able to be installed and to be used for simple use cases in less than 30 minutes. It is not allowed to require further complex installations such as big database systems or other required servers or other systems. If a tool is able to run a simple use case described in the next paragraph it passes this criterion otherwise it fails and is rejected.

The simple use case is as follows:

1. Install the tool
2. Create a sample performance test
3. Send a request to an available web application (for instance www.google.com)

As we have an experienced target group, which is capable of the basics of performance testing and tool usage, we do not expect any threats to validity concerning the different users' skills.

Web Application Testing Ability

As our study aims to test also web applications a tool has to be able to test web applications and offer a simple way to do that. If the tool only offers the possibility to send packets over the net (what basically can test web applications) it is not considered to be able to test web applications in a simple way. The tool has to provide a way to easily define the test case. The usage of third party products or plugins that can help is allowed. If a tool does not offer any of these possibilities it fails this criterion and is rejected.

These criteria represent the must have criteria which have to be fulfilled if a tool should be further taken into account and should participate in further evaluation. Next, the further evaluation criteria categories are presented and the containing criteria are described. These criteria represent no must have criteria but are weighted as described in section 4.1.2. The concrete weighting and value categorization of the criteria can be seen in figure 5.2.

[3] We only take Firefox into account. It is considered a representative browser for all current and modern browsers

5.1.3 General

This category contains the general requirements for a tool. We consider requirements that are not related to usability or performance testing issues but requirements that concern licensing or availability. First criterion of this category is **costs**. We consider the costs of a tool and check if is open source or has a licensing model or cost model. Depending on the costs we rate the tool having open source tools or free tools with the maximum points. The more the tool costs, the less points it gets. This criterion is weighted as critical. Next criterion checks if there is an **User Interface (UI) available** or packaged with the tool or if an UI is at least available for download for the tool. If there is one available the tool gets all points otherwise it does not get points at all for this criterion. This is weighted as high. The **level of actuality** criterion rates the activity of an eventual existing community around the tool and also checks to what extend the project members or community members participate to the project and upcoming questions. Further, the regularity of releases is taken into account here. The more activity can be observed the higher the rating at this criterion is. This is weighted as high. An **integration into development tools** such as Eclipse is been rated with the next criterion. If the tool provides an integration it is rated with full points on this criterion, otherwise with zero points. This is weighted as medium. The last criterion of this category is the **platform support for Mac** platforms. If the tool supports Mac is rated with full points here, otherwise with zero points. This is rated as low.

5.1.4 Usability

With respect to and on base of previous research, such as the characteristics of software quality [21], non-functional requirements [25,65], ISO 9126-1 [52] and a taxonomy of current issues in requirements engineering [78] we consider usability as one category of our criteria and choose non functional criteria as metrics for the evaluation. Usability is an important aspect in terms of future usage of the most promising tool. We want the tool to be helpful when performance testing is needed and we therefore need the tool to be useful. In terms of our evaluation we define seven criteria which we consider to be most relevant for our evaluation. We now describe these criteria.

The first criterion is the **Simplicity of test creation** which expresses the complexity to create more sophisticated test cases than mentioned in “Time to productivity”. This metric reflects the comfort the expert experiences during test creation and learning the tool and implementing test cases. Further, it represents a value that measures if it is easy to create a test case or not. This criterion is rated low if it is complicated relating to the users opinion and it is rated high if the creation is easy. **UI Stability** states if the tool is running without crashing or reporting errors that are not related to a misuse of the tool. We consider a tool as stable if there are no crashes or error reports during the use of the tool and give full points in that case. If there are 1 to 3 crashes we give half of the points and if there are more than 3 crashes a tool gets no point on this criterion. The response time of a tool to user input or other user related actions is also part of the evaluation and we call this criterion **UI Performance**. We measure waiting times in terms of reaction of the users input. If there is a quick response a tool gets full points otherwise there is a deduction of points. We further ask the users if the **UI Intuitivity** and to what extend the users feel comfortable when using the tool. We only consider tools with an UI here. Tools

that do not offer an UI (e.g. those that only offer a command line interface) get no points on this criterion. As a last criterion for this category we introduce the metric of **UI Consistency**. We check whether the tool uses a consistent naming strategy in the whole tool or often changes the names. Further, we check the UI itself for consistent layouting. All the User Interface Metrics are rated by experts of the lab.

5.1.5 Interoperability

As we want the tool to work in an existing development environment we need to express criteria which indicate a tools' interoperability with its environment. Interoperability is an important aspect of tools as there hardly tools that are able to solve all problems by themselves. Therefore, the tools have to inter-operate with other tools or applications. We therefore state some export possibilities. These are listed as follows as rated in an easy way such that if a tool supports an export format or type of export the tool gets full points and zero points otherwise. The export possibilities we check are:

- **Images:** We check whether the tool supports export of results as images.
- **Data:** We check whether the tools is able to export the measured data itself.
- **Formats:** We expect the tool to be able to export XML and/or CSV and/or Unformatted Text data as these are considered as standard formats.
- **Reports:** We check if the tool offers a direct possibility to export full reports about the results.

5.1.6 Performance (Testing) Issues

This category represents all issues that are related with the features of the tool itself. We consider all features that are of great interest for our environment. The resulting criteria are listed as follows: **Supported Systems and Protocols** checks whether a tool is able to support HTTP, HTTPS, JMS, SMTP and FTP as protocol for the performance analysis. Each of the protocols is rated by its own and also has individual weighting. **Handling AJAX requests** is another criteria we define for the tool. As many web applications use the AJAX mechanism we need the tool to support AJAX as well. The rating is full points if the tools supports AJAX or no points otherwise. For the purpose of controlling the amount of load that is applied, we define the **Configurable User Count** criterion. A tool should be able to offer a possibility to configure the amount of users that are generating load. If the tool supports this it gets full points, if there is a possibility that offers a workaround to achieve the user count configuration it gets half the points and otherwise no points. **Load Types** indicates if the load of the users that is generated is fully configurable. Further, we expect a **Support for JUnit Tests**. In the environment there exist acceptance tests that may be useful for performance testing too. As we want to use those in combination with the tool, the tool has to support JUnit tests. Further, for creating tests, we check whether the tool offers a possibility to **Record Test Cases**. This possibility offers a good way to easily create tests cases. If a tool offers a recording possibility it gets full points and no

points otherwise. For observing the running test we further state a criterion that checks the tool for **Live Monitoring**. We expect the tool to offer a way to immediately see the performance after starting the test and live tracking the current performance. If a tool offers live monitoring it gets full points and no points otherwise. Another criterion is the feature of **Distributed Performance Testing**. We want to ensure that we can also test big applications with the tool and distributed testing is a possibility to achieve that. If the tool offers distributed performance testing in gets full points and no points otherwise. **Measureable Metrics** lists metrics that a tool should be able to provide. Each metric is rated a weighted by its own and indicates if the metrics is provided.

5.1.7 Summary

We have now presented all criteria categories and the containing criteria in detail. This answers research issue (RI) 1.1 described in section 3.2 which asks for the requirements of an evaluation study for performance analysis in the research environment. Figure 5.1 sums up the criteria and gives an overview of them[4]. A detailled illustration of the criteria and the weights is shown in 5.2. It maps the criteria to a criteria category and a respective weight.

With the gained user needs and identified criteria, we now proceed with the analysis of the full list of tools (The full list of tools can be seen in Appendix A) by evaluating the must have criteria first and then applying the detailed evaluation.

[4]Criteria marked with an * are split into sub-metrics and can be separated by other sub-weightings and sub-ratings.

	Usabilty	Test/Result Export (Interoperability)	General	Performance (Testing) Issues
Must Have	Time to Productivity	>1 Export Data	Evaluation Availability	Web App Performance Testing possible
			Platform Availability (Win, Linux)	
			No exclusive Cloud Usage	
			Maintenance and Dev Activity	
			Environment Compatibility	
Crititcal	Simplicity of Test Creation	Images	Costs	Supported Systems and Protocols*
		Data		Handling of Ajax Requests
				Measureable Metrics*
				Configureable User Count
				JUnit Support
High	UI Stability	Export Formats*	UI Availability	Recording of Tests
	UI Performance		Level of Actuality	Live Monitoring
Medium	UI Intuitivity		Integration in Dev-Tools	Distributed Testing
	UI Consistency	Automatic Reports		Load Types
Low			Platform Availability (Mac)	

Figure 5.1: Overview of the criteria of the evaluation

	Weighting Factor
Group 1 – General Criteria	
1.1 Costs	7
1.2 UI Availability	4
1.3 Level of actuality (contributions, regularly releases)	4
1.4 Integration in Development Tools	2
1.5 Platform Availability (MacOS)	1
Group 2 – Performance Testing Issues	
2.1 Supported Systems and Protocols	
2.1.1 HTTP	7
2.1.2 HTTPS	7
2.1.3 JMS	2
2.4.4 SMTP	2
2.1.5 FTP	4
2.2 Handling of AJAX-Requests	7
2.3 Load of Users is configurable	7
2.4 Load Types	2
2.5 Support for JUnit Tests	7
2.6 Recording possible	4
2.7 (Live) Monitoring	4
2.8 Distributed Performance Testing	2
2.9 Measureable Metrics	
2.9.1 Response Time	7
2.9.2 Error Count	4
2.9.3 Error Rate	2
2.9.4 Throughput	1
2.9.5 Min Response Time	2
2.9.6 Max Response Time	2
2.9.7 Call Count	2
2.9.8 Mean Response Time	4
Group 3 – Interoperability	
3.1 Image Create/Export	7
3.2 Data Export	7
3.3 Export Formats	
3.3.1 XML	4
3.3.2 CSV	2
3.3.3 Unformatted Text File	2
3.4 Automatic Report Creation	2
Group 4 – Usability	
4.1 Simplicity of Test Creation	7
4.2 UI Stability	4
4.3 UI Performance	4
4.4 UI Intuitivity	2
4.5 UI Consistency	2

Figure 5.2: Criteria Categorization and Weightings

5.2 Tool Evaluation

To find a list that is very complete and contains all available tools, we search for available tools on multiple search engines. We can retrieve many tools directly by reading the results of the search engines. Some tools are a little more hidden and are thus found over summary sites that summarize a category of tools. We merge all results and create the full list of available tools out of all subsequent findings of all resources (A full list of the tools and the resources can be seen in Appendix A. The evaluation of the must have criteria mentioned in the section above leads to the short list of tools. With this short list we continue with the further study. The detailed results of the evaluation of the must have criteria can be viewed in table 5.1.

ID	Toolname	Version	Evaluation Availability	Platform Availabilty (Windows/Linux)	No exclusive cloud usage	Maintenance and dev activity	Environment Compatibility	Data Export Possibility	Time to Productivity	Web Application Testing Ability	Detailled Investigation
1	Allmon	0.2.0-PA	yes	yes	yes	no	yes	yes	no	no	N
2	Oracle ATS [5]	12.4.0.2.0	yes	yes	yes	yes	yes	yes	no	yes	N
3	Appvance	-	no	N/A	no	N/A	N/A	yes	N/A	yes	N
4	BFS[6]	-	no	N/A	no	N/A	N/A	no	N/A	yes	N
5	benerator	0.9.7	yes	yes	yes	yes	yes	no	yes	no	N
6	BlazeMeter	-	no	N/A	no	N/A	N/A	N/A	N/A	yes	N
7	Blitz	-	no	N/A	no	N/A	N/A	yes	N/A	yes	N
8	CitraTest	-	no	N/A	N/A	N/A	N/A	N/A	N/A	yes	N
9	CLIF	2.2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes
10	CloudTest	-	no	N/A	no	N/A	N/A	N/A	N/A	yes	N
11	contiperf	2.3.4	yes	yes	yes	yes	yes	yes	yes	yes	yes
12	Curl-loader	0.56	no	yes	yes	yes	no	no	no	yes	N
13	D-ITG	2.8.1	yes	yes	yes	yes	yes	no	no	no	N
14	DOTS[7]	-	no	no	yes	yes	yes	no	no	no	N
15	DBMonster	1.0.3	yes	yes	yes	yes	yes	no	no	no	N
16	Deluge	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N

[5] Application Testing Suite
[6] Benchmark Factory Suite
[7] Database Open Test Suite

ID	Toolname	Version	Evaluation Availability	Platform Availabilty (Windows/Linux)	No exclusive cloud usage	Maintenance and dev activity	Environment Compatibility	Data Export Possibility	Time to Productivity	Web Application Testing Ability	Detailled Investigation
17	Dieseltest	1.0.21	yes	yes	yes	no	yes	N/A	N/A	yes	N
18	EclipseProfiler	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
19	EJP	2.2beta1	yes	yes	yes	no	yes	N/A	yes	no	N
20	Faban	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
21	Funkload	1.17.0[8]	yes	yes	yes	yes	no	yes	N/A	yes	N
22	FWPTT	0.7	yes	yes	yes	no	yes	yes	yes	yes	N
23	Gatling	2.0.3	yes	yes	yes	yes	no	yes	yes	yes	N
24	Grinder	3.11	yes	yes	yes	yes	yes	yes	yes	yes	yes
25	Grinderstone	2.5.5	yes	yes	yes	no	yes	yes	yes	yes	N
26	HammerDB	2.16	yes	yes	yes	yes	yes	yes	yes	no	N
27	Hammerhead 2	2.1.4	yes	yes	yes	no	no	N/A	N/A	yes	N
28	HP Load Runner	12.02	yes	yes	yes	yes	no	yes	yes	yes	N
29	http_load	14aug2014	yes	yes	yes	N/A	no	yes	yes	yes	N
30	httperf	0.9.0	yes	yes	yes	no	yes	no	yes	yes	N
31	Eclipse Hyades	4.7.2	yes	yes	yes	no	yes	N/A	yes	yes	N
32	IPerf	3.0.9	yes	yes	yes	yes	yes	yes	yes	N/A	N
33	IxoraRMS	1.1	yes	yes	yes	no	yes	yes	yes	yes	N
34	j-hawk	11.5	yes	yes	yes	yes	yes	N/A	yes	no	N
35	jchav	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
36	JCrawler	1.5beta	yes	yes	yes	no	yes	yes	yes	yes	N
37	JMeter	2.12	yes	yes	yes	yes	yes	yes	yes	yes	yes
38	Jprobe	8.0	no	yes	yes	no	yes	N/A	yes	no	N
39	jProf	-	no	N/A	yes	N/A	N/A	N/A	N/A	no	N
40	Jstress	0.50	yes	yes	yes	no	yes	N/A	N/A	yes	N
41	JUnit Perf	1.9	yes	yes	yes	no	yes	yes	yes	yes	N
42	JUnit Scenario	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
43	Load Impact	-	no	N/A	no	N/A	N/A	N/A	N/A	N/A	N
44	Load Storm	-	no	N/A	no	N/A	N/A	N/A	N/A	N/A	N

[8] b-20140310

ID	Toolname	Version	Evaluation Availability	Platform Availabilty (Windows/Linux)	No exclusive cloud usage	Maintenance and dev activity	Environment Compatibility	Data Export Possibility	Time to Productivity	Web Application Testing Ability	Detailled Investigation
45	loader.io	-	no	N/A	no	N/A	N/A	N/A	N/A	N/A	N
46	LoadSim	0.9.8	yes	yes	yes	no	yes	N/A	N/A	yes	N
47	Loadster	-	no	N/A	no	N/A	N/A	N/A	N/A	N/A	N
48	LoadComplete	2.8	yes	yes	yes	yes	yes	yes	yes	yes	yes
49	Lobo	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
50	LoginVSI	-	yes	yes	N/A	N/A	N/A	N/A	N/A	no	N
51	Messadmin	5.4	yes	yes	yes	no	yes	N/A	N/A	no	N
52	MStone	4.9.4	yes	yes	yes	no	yes	N/A	N/A	no	N
53	Multi-Mechanize	1.2.0	yes	yes	yes	no	no	N/A	N/A	yes	N
54	NeoLoad	5.0.2	yes	yes	yes	yes	yes	yes	yes	yes	yes
55	NGrinder	3.3	yes	yes	yes	yes	yes	yes	yes	yes	N
56	NTime	-	yes	yes	yes	no	no	N/A	N/A	yes	N
57	OpenSTA	1.4.4	yes	yes	yes	no	yes	N/A	N/A	yes	N
58	OpenWebLoad	0.1.2	yes	yes	yes	no	N/A	N/A	N/A	yes	N
59	OptimizeIt	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
60	Ostinato	0.6	yes	yes	yes	yes	yes	N/A	N/A	no	N
61	P-unit	0.15	yes	yes	yes	no	yes	yes	N/A	yes	N
62	P6Spy	1.3	yes	yes	yes	no	yes	N/A	N/A	no	N
63	PandoraFMS	5.1	no	yes	yes	yes	yes	N/A	N/A	no	N
64	PerfectLoad	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
65	PerformaSure	TKU[9]	no	N/A	N/A	N/A	N/A	N/A	N/A	no	N
66	postal	-	no	N/A	N/A	N/A	N/A	N/A	N/A	no	N
67	Pylot	1.26	yes	yes	yes	no	no	N/A	N/A	yes	N
68	QACenter	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
69	QEngine	-	no	N/A	N/A	no	N/A	N/A	N/A	N/A	N
70	Rational[10]	-	yes	yes	yes	yes	yes	yes	no	yes	N
71	Raw Load Tester	v1.0	yes	yes	yes	no	no	no	yes	yes	N

[9] 2010-Jun-1

[10] Performance Tester

ID	Toolname	Version	Evaluation Availability	Platform Availabilty (Windows/Linux)	No exclusive cloud usage	Maintenance and dev activity	Environment Compatibility	Data Export Possibility	Time to Productivity	Web Application Testing Ability	Detailled Investigation
72	Seagull	1.8.2	yes	yes	yes	no	yes	yes	N/A	no	N
73	Siege	3.0.8	yes	yes	yes	yes	no	N/A	N/A	yes	N
74	SilkPerformer	15.5	yes	yes	yes	yes	yes	yes	yes	yes	yes
75	SimpleProfiler	-	no	N/A	yes	no	N/A	N/A	N/A	no	N
76	Sipp	3.4	yes	N/A	yes	yes	N/A	N/A	N/A	no	N
77	SLAMD	-	no	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
78	Soap-Stone	0.952	yes	yes	yes	no	yes	yes	yes	no	N
79	SOATest	9.8	yes	N/A	yes	N/A	N/A	N/A	N/A	no	N
80	stress_driver	1.0	yes	N/A	yes	no	no	N/A	N/A	yes	N
81	Test Load	-	no	N/A	N/A	N/A	N/A	N/A	N/A	yes	N
82	TestComplete	-	yes	yes	yes	yes	yes	yes	yes	no	N
83	Testing Anywhere	9.2	yes	yes	yes	yes	yes	yes	yes	yes	yes
84	TestMaker	6.1	yes	yes	yes	N/A	yes	N/A	N/A	N/A	N
85	TestStudio	Ultimate[11]	yes	yes	yes	yes	yes	yes	yes	yes	yes
86	tivoli[12]	-	no	N/A	no	N/A	N/A	N/A	N/A	N/A	N
87	TPTEST	5_0_2	yes	yes	yes	no	yes	N/A	N/A	no	N
88	Tsung	1.5.1	yes	N/A	yes	yes	no	N/A	N/A	yes	N
89	Valgrind	3.10.1	yes	no	yes	yes	no	N/A	N/A	no	N
90	Visual Studio	2013[13]	yes	yes	yes	yes	no	yes	yes	yes	N
91	WAPT	Pro 3.5	yes	yes	yes	yes	yes	yes	yes	yes	yes
92	Web Polygraph	4.3.2	yes	N/A	yes	N/A	N/A	N/A	N/A	no	N
93	WebLOAD	10.2	yes	yes	yes	yes	yes	yes	no	yes	N
94	App Perfect	-	yes	yes	yes	N/A	yes	N/A	N/A	yes	N
95	Apache Flood	-	yes	yes	yes	N/A	yes	N/A	N/A	no	N

Table 5.1: Must-Have Evaluation

[11] 2014_4_1211
[12] monitoring/composite
[13] update4

According to the full list, we get the following tools that remain after the must have evaluation:

ID	Tool Name	Tool Vendor	Version
9	CLIF	OW2	2.2.1
11	contiperf	databene	2.3.4
24	Grinder	Paco Gomez, Philip Aston	3.11
37	JMeter	Apache Software Foundation	2.12
48	LoadComplete	smartbear	2.8
54	NeoLoad	Neotys	5.0.2
74	SilkPerformer	Borland	15.5
83	Testing Anywhere	Automation Anywhere	9.2
85	TestStudio	Telerik	Ultimate 2014_4_1211
91	WAPT	SoftLogica	Pro 3.5

Table 5.2: List of remaining tools after the must have criteria selection

These tools pass the test that the must have criteria must be fulfilled. For detailed information about the evaluation results, please see table 5.1. We could restrict the detailed evaluation from originally 95 tools to 10 tools with this evaluation. That is 10,53%.

In the following we take a look into each tool and present the results of the detailed tool evaluation. We present the tools in detail and describe the advantages as well as the drawbacks of each of the tools. We also give a short idea how the user interface looks like. For a detailed evaluation data see section 5.2.11.

5.2.1 CLIF

The CLIF Framework[14] from the OW2 Consortium is an open load testing platform. The GUI is based either on Eclipse or Swing. Figure 5.3 shows the Eclipse UI which we used for our needs.

Further, there is a Command Line Interface and a plugin for the common CI-Server Hudson/-Jenkins. It provides so called “Injectors” which are related to a technology that should be tested. Out of the box, CLIF provides protocols, such as HTTP/HTTPS or FTP and offers injectors for that. Further, it provides so called “Probes” for measuring metrics.

The CLIF Framework performed well with some features missing. For example, we could not find a SMTP Injector and also there is no possibility to run JUnit Tests within the tool. Distributed Performance Testing on many clients was also not detected. Also the rating of the user interface category was not rated with full points by the experts due to a little complexity in the test creation and intuitivity.

Compared to the most fitting tool, CLIF evaluated good with several minor issues that led to deduction of points.

[14]CLIF: http://clif.ow2.org/

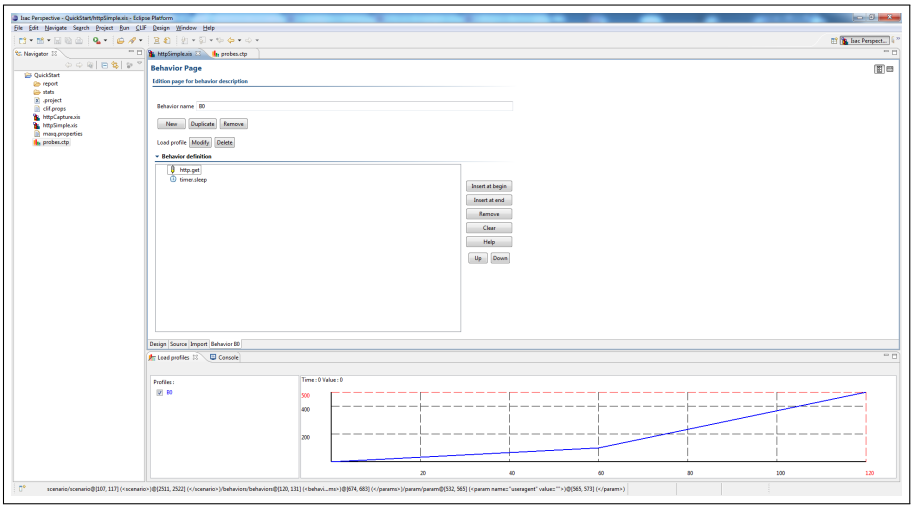


Figure 5.3: CLIF Overview

	General	Performance Testing Issues	Interoperability	Usability
CLIF	100,00%	83,54%	52,08%	79,21%

Table 5.3: Final Result of the Evaluation - CLIF

5.2.2 contiperf

Contiperf[15] follows a slightly different approach than most of the other evaluated tools. It is only an extension for the common Unit Testing Framework JUnit that extends the functionality of JUnit with performance testing features. Some of the features are:

1. Repeating tests with exact invocation and thread count
2. Defining of SLA similar restrictions
3. Providing warm-up and ramp-up periods
4. Parallel test execution
5. Integration with Maven

There are many more features that can be seen on the tool homepage. Due to the completely different approach contiperf did not perform well on our set of criteria.

To see how tests are implemented using contiperf, we present a simple tests that is implemented with this tool. Listing 5.1 shows the code:

```
import org.junit.*;
import org.databene.contiperf.*;
```

[15] contiperf: http://databene.org/contiperf

	General	Performance Testing Issues	Interoperability	Usability
Contiperf	77,78%	29,11%	45,83%	36,84%

Table 5.4: Final Result of the Evaluation - contiperf

```
public class SmokeTest {
        @Rule
        public ContiPerfRule i = new ContiPerfRule();

        @Test
        @PerfTest(invocations = 1000, threads = 20)
        @Required(max = 1200, average = 250)
        public void test1() throws Exception {
                Thread.sleep(200);
        }
}
```

Listing 5.1: Contiperf example taken from http://databene.org/contiperf

5.2.3 Grinder

Grinder[16] is a load testing framework that is basically able to test all components of software that offer a Java API. This includes of course HTTP components such as web applications or other components such as CORBA servers. Grinder offers a scripting possibility to express tests within languages such as Java, Jython or Clojure. An example of a test can be seen in listing 5.2 The architecture of Grinder splits up two important parts of the framework: Controlling the tests and executing the tests. Figure 5.4 provides a introduction of the GUI.

Therefore, it offers a console and an agent process that implement those two parts. It is designated to be able to run the agents on different machines to achieve a higher load and perform distributed performance testing.

```
# A simple example using the HTTP plugin that shows the
# retrieval of a single page via HTTP.

from net.grinder.script import Test
from net.grinder.script.Grinder import grinder
from net.grinder.plugin.http import HTTPRequest

test1 = Test(1, "Request␣resource")
request1 = HTTPRequest()
test1.record(request1)
```

[16]Grinder: http://grinder.sourceforge.net/

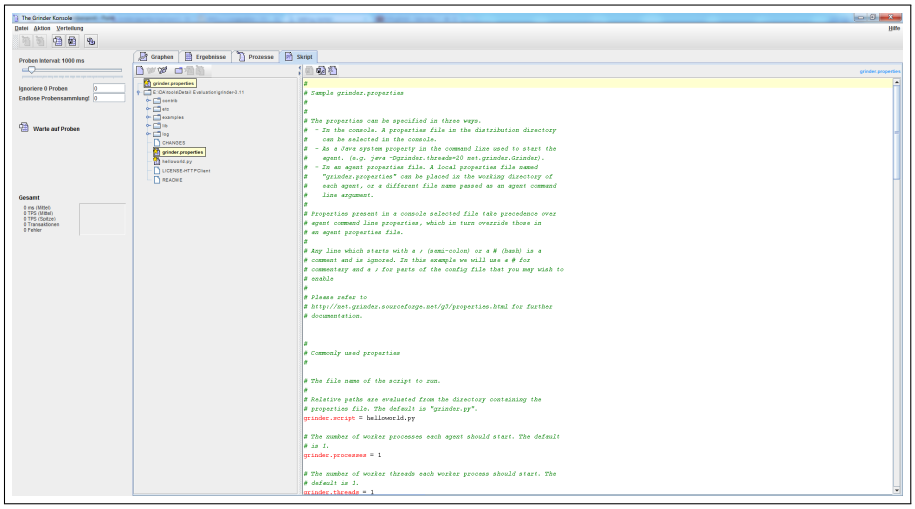


Figure 5.4: Grinder Overview

```
class TestRunner:
    def __call__(self):
        result = request1.GET("http://localhost:7001/")
```

Listing 5.2: Grinder test example taken from http://grinder.sourceforge.net/g3/scripts.html

Grinder does not offer a definition of load types but that can be handled with the agents themselves. It further does not provide all requested metrics out of the box and the export formats are restricted. In terms of usability there a minor problems that one has to be familiar with, such as configuring applications via configuration files and the GUI itself is a little bit confusing concerning the data that is distributed to run the tests and the test execution.

	General	Performance Testing Issues	Interoperability	Usability
Grinder	88,89%	86,08%	37,50%	55,26%

Table 5.5: Final Result of the Evaluation - Grinder

5.2.4 LoadComplete

LoadComplete[17] is a load testing framework that aims to keep the users focus on the performance testing tasks rather than on programming tasks. The vendor, smartbear, claims that with LoadComplete it does not require programming knowledge to use LoadComplete. Figure 5.5 shows the GUI of LoadComplete.

[17] LoadComplete: http://smartbear.com/products/qa-tools/load-testing/

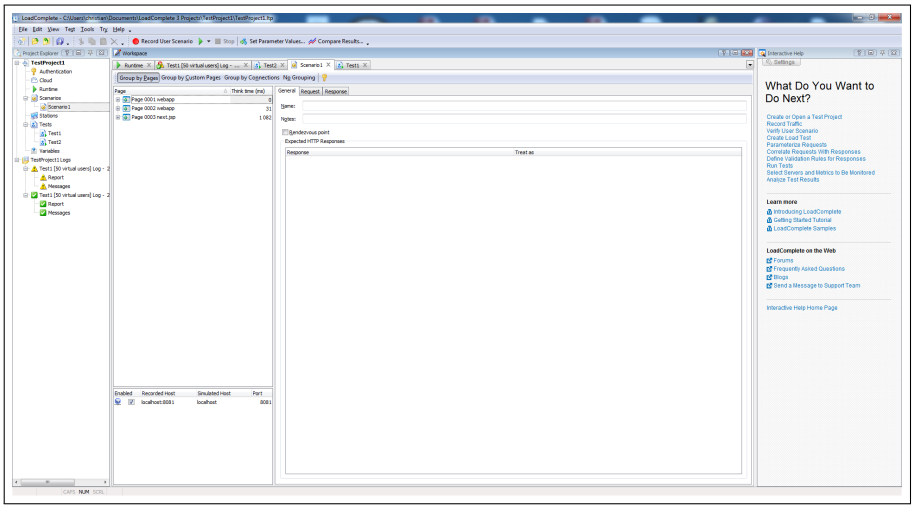


Figure 5.5: LoadComplete Overview

While using LoadComplete we could not detect JMS, SMTP or FTP support out of the box. Further, the way how to define the user load type is not optimal and the experts found some flaws in the user interface intuitivity. Due to this, the simplicity of test creation suffered.

	General	Performance Testing Issues	Interoperability	Usability
LoadComplete	50,00%	75,95%	62,50%	76,32%

Table 5.6: Final Result of the Evaluation - LoadComplete

5.2.5 NeoLoad

NeoLoad[18] is a load framework specialized for web and mobile applications provided by NEOTYS. Main features of the tool are scriptfree implementation of tests, recording of tests or the mobile testing possibility. NeoLoad is delivered with a GUI out of the box that allows to compose test cases by using the predefined elements.

The user load is free configurable through a graph and also predefined load types are available. Similar to some other tools there is a lack of JMS, SMTP and FTP support in NeoLoad most probably because it is specialized to web applications. Also the experts rated the user interface as a little unintuitive due to many tabs and sub tabs.

	General	Performance Testing Issues	Interoperability	Usability
NeoLoad	33,33%	78,48%	91,67%	76,32%

Table 5.7: Final Result of the Evaluation - NeoLoad

[18]NeoLoad: http://www.neotys.de/product/overview-neoload.html

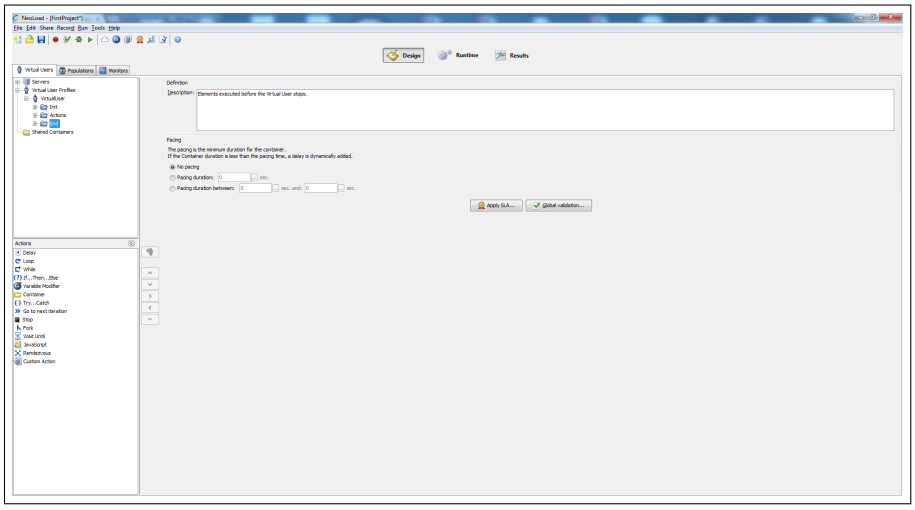


Figure 5.6: NeoLoad Overview

5.2.6 SilkPerformer

SilkPerformer[19] is a commercial tool developed by Borland. It supports many protocols and web technologies such as, HTTP/HTTPS, AJAX, Flash/Flex or Silverlight. Borland itself claims SilkPerformer to be "the world leader in performance testing". It has a very extensive reporting feature and a list of predefined user load types. There is a possibility of arranging load agents. That leads to the possibility of distributed performance testing.

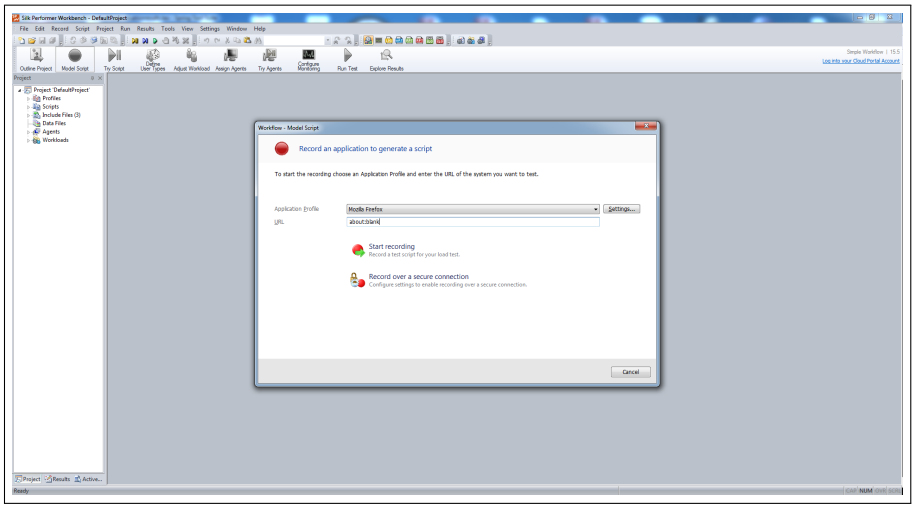


Figure 5.7: SilkPerformer Overview

It also seems as there are less but bigger releases for the tool. Further, the experts state that

[19] SilkPerformer: http://www.borland.com/products/silkperformer/

the user interface appears a little bit complicated and compared to other tools it is more difficult to create simple tests.

	General	Performance Testing Issues	Interoperability	Usability
SilkPerformer	44,44%	98,73%	83,33%	76,32%

Table 5.8: Final Result of the Evaluation - SilkPerformer

5.2.7 Testing Anywhere

Testing Anywhere[20] is a load testing framework that provides a visual load test editor. The editor supports the recording of the tests and provides a possibility to parametrize tests to repeat tests with different parameters. The editor further offers an interface to edit recorded tests and organizes the load tests.

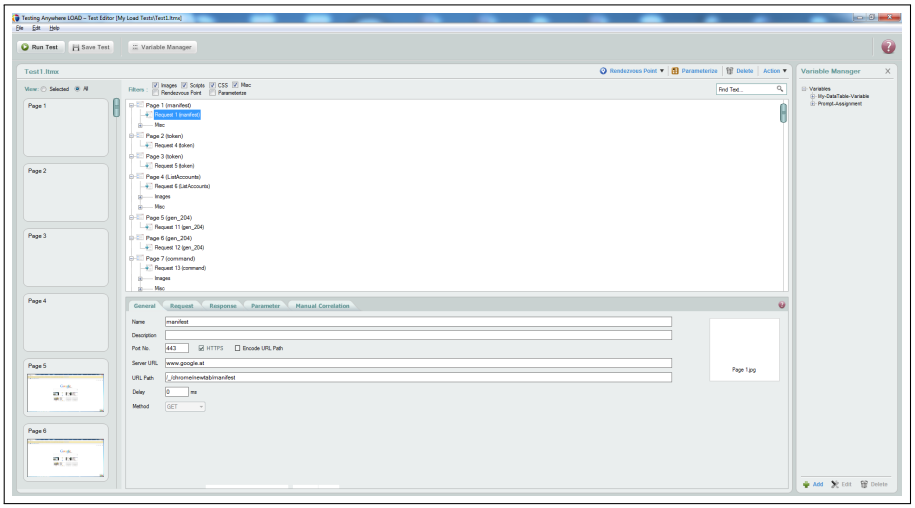


Figure 5.8: Testing Anywhere Overview

During the test of the tool we encountered a crash of the tool for no reason so there is a drawback at this criterion. Further, the tool did not perform well in terms of export formats but it offers a possibility to generate reports. The tool further does not provide an IDE integration.

	General	Performance Testing Issues	Interoperability	Usability
Testing Anywhere	44,44%	79,75%	52,08%	89,47%

Table 5.9: Final Result of the Evaluation - Testing Anywhere

[20]Testing Anywhere: http://www.automationanywhere.com/Testing/

5.2.8 TestStudio

During the evaluation of Test Studio[21] we encountered several crashes while trying to record a test. Therefore, we could not create a performance test and verify the rest of the tool. Further, the usability was rated low of this tool because of the complicated way to start tests. Hence, we could not do a deeper study. Figure 5.9 presents the user interface of the tool.

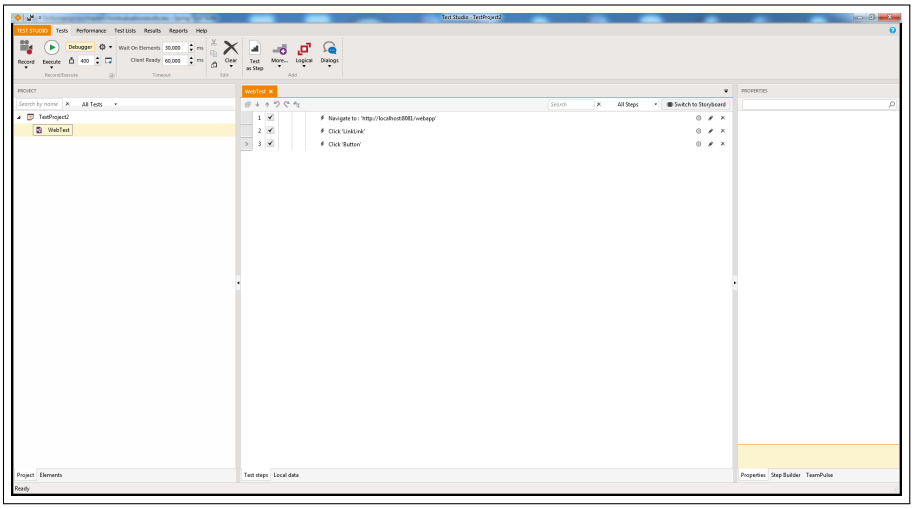


Figure 5.9: Test Studio Overview

5.2.9 WAPT

WAPT[22] is a performance testing framework by SoftLogica. It provides an assistant for creating tests which enables the user to quickly get to tests. It offers a possibility to distribute the load on different agents. The user load can be set with predefined values ans load types and is displayed in a graph for better understanding.

WAPT offers various reporting mechanisms that can also be exported as HTML reports or CSV reports. The results are further presented as graphs within the tool user interface. WAPT was not able to test JMS, FTP or SMTP applications and did not support JUnit tests.

	General	Performance Testing Issues	Interoperability	Usability
WAPT	33,33%	79,75%	83,33%	100,00%

Table 5.10: Final Result of the Evaluation - WAPT

[21] TestStudio: http://www.telerik.com/teststudio
[22] http://www.loadtestingtool.com/index.shtml

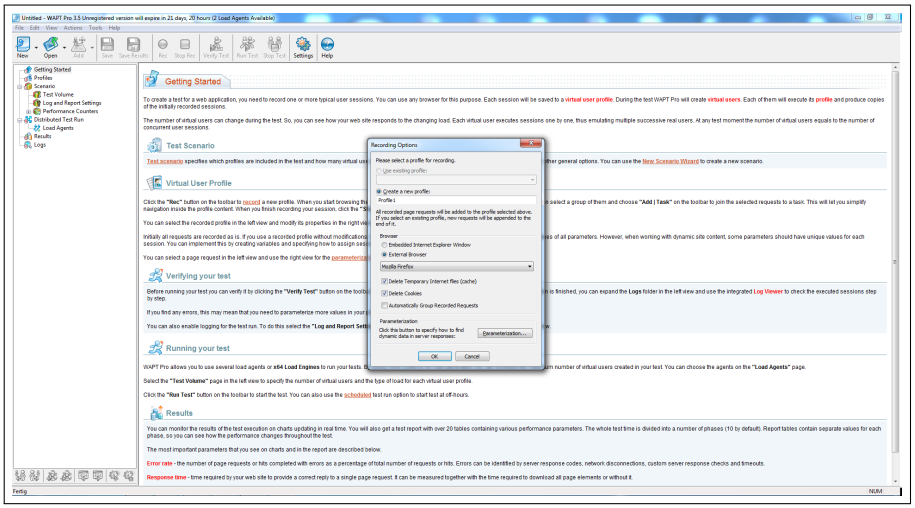


Figure 5.10: WAPT Overview

5.2.10 JMeter

JMeter[23] is an open source performance and load test framework developed by the Apache Software Foundation[24]. It was originally designed to test web application but has evolved to a multi purpose performance testing tool. It supports many formats and protocols, such as HTTP/HTTPS, SMTP, JMS, FTP or LDAP. Due to the fact that JMeter is completely written in Java, it is 100% portable to all operating systems that support Java and for those that provide a Java Runtime Environment. JMeter is shipped with a GUI out of the box. Figure 5.11 shows the basic user interface.

JMeter is highly extensible because of the plugin mechanisms. One can implement an own sampler or listener or other plugins and extend the tool with it. The documentation on the tool homepage is detailed and supports the users with the necessary information to get started and even for further more complicated scenarios. JMeter has no explicit automatic report generation out of the box but offers plugins for composing data and graphs and for exporting the composition in various formats as well as graphics.

	General	Performance Testing Issues	Interoperability	Usability
JMeter	88,89%	98,73%	83,33%	100,00%

Table 5.11: Final Result of the Evaluation - JMeter

[23] JMeter: http://jmeter.apache.org/
[24] http://www.apache.org/

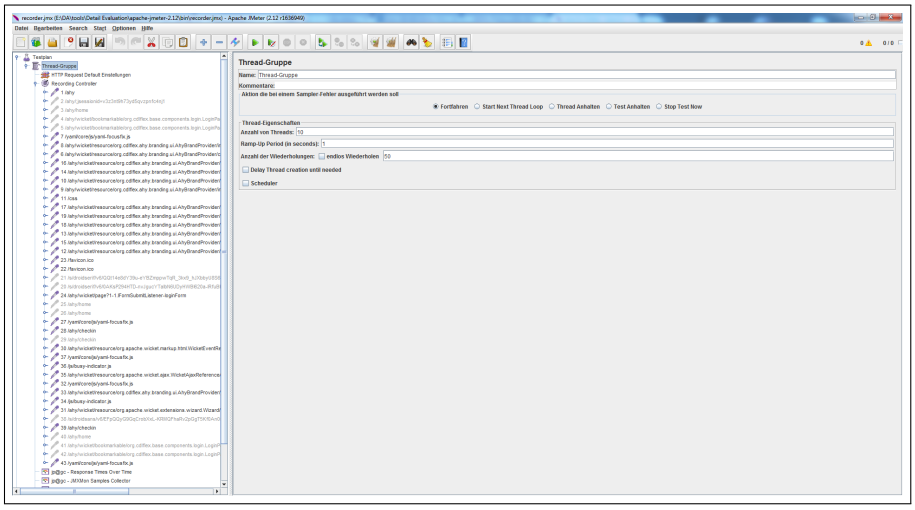


Figure 5.11: JMeter Overview

5.2.11 Evaluation Summary and Most Fitting Tool

In this we present the complete detailed evaluation results. The full data is listed in tables 5.12, 5.13 and 5.14

As we can see in the detailed evaluation data, the sum of the points lead to a ranking of the evaluated tool ordering from the most fitting tool to the least fitting tool for our research environment and requirements. We split up the visualization in the four main categories and the results are presented in table 5.12. This table presents the scores of each of the tools in each of the four categories. 100,00% means that the respective category is completely fulfilled by the respective tool.

ID	Tool Name	General	Performance Testing Issues	Interoperability	Usability
9	CLIF	100,00%	83,54%	52,08%	79,21%
37	JMeter	88,89%	98,73%	83,33%	100,00%
11	Contiperf	77,78%	29,11%	45,83%	36,84%
24	Grinder	88,89%	86,08%	37,50%	55,26%
91	WAPT	33,33%	79,75%	83,33%	100,00%
54	NeoLoad	33,33%	78,48%	91,67%	76,32%
74	SilkPerformer	44,44%	98,73%	83,33%	76,32%
48	LoadComplete	50,00%	75,95%	62,50%	76,32%
83	Testing Anywhere	44,44%	79,75%	52,08%	89,47%
85	TestStudio	-	-	-	-

Table 5.12: Final Result of the Evaluation - Scores by Category

This leads to a final result presented in table 5.13. It sums up the results of the main categories and shows the overall evaluation results. The column “Achieved Points” reflects the

	Weighting Factor	CLIF		JMeter		Contiperf	
		Rate	Score	Rate	Score	Rate	Score
Group 1 – General Criteria							
1.1 Costs	7	1	7	1	7	1	7
1.2 UI Availability	4	1	4	1	4	0,5	2
1.3 Level of actuality	4	1	4	1	4	0,5	2
1.4 Integration in Development Tools	2	1	2	0	0	1	2
1.5 Platform Availability (MacOS)	1	1	1	1	1	1	1
Group 2 – Performance Testing Issues							
2.1 Supported Systems and Protocols							
2.1.1 HTTP	7	1	7	1	7	0	0
2.1.2 HTTPS	7	1	7	1	7	0	0
2.1.3 JMS	2	1	2	1	2	0	0
2.4.4 SMTP	2	0	0	1	2	0	0
2.1.5 FTP	4	1	4	1	4	0	0
2.2 Handling of AJAX-Requests	7	1	7	1	7	0	0
2.3 Load of Users is configurable	7	1	7	1	7	1	7
2.4 Load Types	2	1	2	0,5	1	0,5	1
2.5 Support for JUnit Tests	7	0	0	1	7	1	7
2.6 Recording possible	4	1	4	1	4	0	0
2.7 (Live) Monitoring	4	0,5	2	1	4	0	0
2.8 Distributed Performance Testing	2	0	0	1	2	0	0
2.9 Measureable Metrics							
2.9.1 Response Time	7	1	7	1	7	0	0
2.9.2 Error Count	4	1	4	1	4	0	0
2.9.3 Error Rate	2	1	2	1	2	0	0
2.9.4 Throughput	1	1	1	1	1	0	0
2.9.5 Min Response Time	2	1	2	1	2	0	0
2.9.6 Max Response Time	2	1	2	1	2	1	2
2.9.7 Call Count	2	1	2	1	2	1	2
2.9.8 Mean Response Time	4	1	4	1	4	1	4
Group 3 – Interoperability							
3.1 Image Create/Export	7	0,5	3,5	1	7	0	0
3.2 Data Export	7	1	7	1	7	1	7
3.3 Export Formats							
3.3.1 XML	4	0	0	1	4	1	4
3.3.2 CSV	2	0	0	1	2	0	0
3.3.3 Unformatted Text File	2	1	2	0	0	0	0
3.4 Automatic Report Creation	2	0	0	0	0	0	0
Group 4 – Usability							
4.1 Simplicity of Test Creation	7	0,75	5,25	1	7	1	7
4.2 UI Stability	4	1	4	1	4	0	0
4.3 UI Performance	4	0,8	3,2	1	4	0	0
4.4 UI Intuitivity	2	0,3	0,6	1	2	0	0
4.5 UI Consistency	2	1	2	1	2	0	0
		Total:	**111,55**	**Total:**	**133**	**Total:**	**55**

Figure 5.12: Part 1 of the Detailed Evaluation Results

	Weighting Factor	Grinder		WAPT		NeoLoad	
		Rate	Score	Rate	Score	Rate	Score
Group 1 – General Criteria							
1.1 Costs	7	1	7	0	0	0	0
1.2 UI Availability	4	1	4	1	4	1	4
1.3 Level of actuality	4	0,5	2	0,5	2	0,5	2
1.4 Integration in Development Tools	2	1	2	0	0	0	0
1.5 Platform Availability (MacOS)	1	1	1	0	0	0	0
Group 2 – Performance Testing Issues							
2.1 Supported Systems and Protocols							
2.1.1 HTTP	7	1	7	1	7	1	7
2.1.2 HTTPS	7	1	7	1	7	1	7
2.1.3 JMS	2	1	2	0	0	0	0
2.4.4 SMTP	2	1	2	0	0	0	0
2.1.5 FTP	4	1	4	0	0	0	0
2.2 Handling of AJAX-Requests	7	1	7	1	7	1	7
2.3 Load of Users is configurable	7	1	7	1	7	1	7
2.4 Load Types	2	0	0	0,5	1	1	2
2.5 Support for JUnit Tests	7	1	7	0	0	0	0
2.6 Recording possible	4	1	4	1	4	1	4
2.7 (Live) Monitoring	4	1	4	1	4	1	4
2.8 Distributed Performance Testing	2	1	2	1	2	1	2
2.9 Measureable Metrics							
2.9.1 Response Time	7	1	7	1	7	1	7
2.9.2 Error Count	4	1	4	1	4	1	4
2.9.3 Error Rate	2	0	0	1	2	1	2
2.9.4 Throughput	1	0	0	1	1	1	1
2.9.5 Min Response Time	2	0	0	1	2	1	2
2.9.6 Max Response Time	2	0	0	1	2	1	2
2.9.7 Call Count	2	0	0	1	2	0	0
2.9.8 Mean Response Time	4	1	4	1	4	1	4
Group 3 – Interoperability							
3.1 Image Create/Export	7	0	0	1	7	1	7
3.2 Data Export	7	1	7	1	7	1	7
3.3 Export Formats							
3.3.1 XML	4	0	0	0	0	1	4
3.3.2 CSV	2	0	0	1	2	1	2
3.3.3 Unformatted Text File	2	1	2	1	2	0	0
3.4 Automatic Report Creation	2	0	0	1	2	1	2
Group 4 – Usability							
4.1 Simplicity of Test Creation	7	0,5	3,5	1	7	0,5	3,5
4.2 UI Stability	4	0,5	2	1	4	1	4
4.3 UI Performance	4	1	4	1	4	1	4
4.4 UI Intuitivity	2	0,25	0,5	1	2	0,5	1
4.5 UI Consistency	2	0,25	0,5	1	2	1	2
		Total:	**103,5**	**Total:**	**108**	**Total:**	**104,5**

Figure 5.13: Part 2 of the Detailed Evaluation Results

	Weighting Factor	SilkPerfomer		LoadComplete		Testing Anywhere	
		Rate	Score	Rate	Score	Rate	Score
Group 1 – General Criteria							
1.1 Costs	7	0	0	0	0	0	0
1.2 UI Availability	4	1	4	1	4	1	4
1.3 Level of actuality	4	0,5	2	1	4	1	4
1.4 Integration in Development Tools	2	1	2	0	0	0	0
1.5 Platform Availability (MacOS)	1	0	0	1	1	0	0
Group 2 – Performance Testing Issues							
2.1 Supported Systems and Protocols							
2.1.1 HTTP	7	1	7	1	7	1	7
2.1.2 HTTPS	7	1	7	1	7	1	7
2.1.3 JMS	2	1	2	0	0	0	0
2.4.4 SMTP	2	1	2	0	0	0	0
2.1.5 FTP	4	1	4	0	0	0	0
2.2 Handling of AJAX-Requests	7	1	7	1	7	1	7
2.3 Load of Users is configurable	7	1	7	1	7	1	7
2.4 Load Types	2	0,5	1	0,5	1	0,5	1
2.5 Support for JUnit Tests	7	1	7	0	0	0	0
2.6 Recording possible	4	1	4	1	4	1	4
2.7 (Live) Monitoring	4	1	4	1	4	1	4
2.8 Distributed Performance Testing	2	1	2	1	2	1	2
2.9 Measureable Metrics							
2.9.1 Response Time	7	1	7	1	7	1	7
2.9.2 Error Count	4	1	4	1	4	1	4
2.9.3 Error Rate	2	1	2	0	0	1	2
2.9.4 Throughput	1	1	1	0	0	1	1
2.9.5 Min Response Time	2	1	2	1	2	1	2
2.9.6 Max Response Time	2	1	2	1	2	1	2
2.9.7 Call Count	2	1	2	1	2	1	2
2.9.8 Mean Response Time	4	1	4	1	4	1	4
Group 3 – Interoperability							
3.1 Image Create/Export	7	1	7	0	0	0,5	3,5
3.2 Data Export	7	1	7	1	7	1	7
3.3 Export Formats							
3.3.1 XML	4	1	4	1	4	0	0
3.3.2 CSV	2	0	0	1	2	0	0
3.3.3 Unformatted Text File	2	0	0	0	0	0	0
3.4 Automatic Report Creation	2	1	2	1	2	1	2
Group 4 – Usability							
4.1 Simplicity of Test Creation	7	0,5	3,5	0,5	3,5	1	7
4.2 UI Stability	4	1	4	1	4	0,5	2
4.3 UI Performance	4	1	4	1	4	1	4
4.4 UI Intuitivity	2	0,5	1	0,5	1	1	2
4.5 UI Consistency	2	1	2	1	2	1	2
		Total:	**120,5**	**Total:**	**98,5**	**Total:**	**100,5**

Figure 5.14: Part 3 of the Detailed Evaluation Results

points each tool got during the evaluation phase. The column “Achieved Score” reflects the score related to the maximum possible score.

ID	Tool Name	Achieved Points (rounded)	Achieved Score (rounded)
37	JMeter	133	95%
74	SilkPerformer	121	86%
9	CLIF	112	80%
91	WAPT	108	77%
54	NeoLoad	105	75%
24	Grinder	104	74%
83	Testing Anywhere	101	72%
48	LoadComplete	99	70%
11	Contiperf	55	39%
85	TestStudio	-	-

Table 5.13: Final Result of the Evaluation - Ranking of the Tools

Concerning the research questions stated in section 3.2, tables 5.12 and 5.13 also present answers to RI 1.2 and RI 1.3. Regarding RI 1.2 we can see the amount of fulfillment of the requirements of each tool in table 5.12 and see to what extent each tool fulfills the respective categories. Regarding RI 1.3 we can see in table 5.13 the overall score of the tools and we can further conclude that for our research environment and the subsequent requirements JMeter is the best choice.

CHAPTER 6

Test Framework Design and Implementation

In this chapter we present the results of the test design and the relating implementation. We use the most promising tool - JMeter - which resulted from the evaluation study of chapter 5. We further use the solution approach described in section 4.2 and the proposed tests. We go through each test and present the design and the implementation.

6.1 General Design

Before the actual tests can be designed and implemented, some potential pitfalls and respective solutions have to be discussed. These problems contain topics that are related to the web application or technical implications out of that or implications of the usage of JMeter. We discuss the following topics:

- **User Load Controlling** In order to be able to control the load that is applied to the system, we have to take care of the user count that is used.
- **Recording of Test Cases** We want to record the testcases to be as precise as possible concerning requests.
- **Session Handling** Session Handling can be a tricky part of any web application testing as we need to be logged in during the test.
- **Handling of Files and Paths** The use case contains a file upload mechanism which we test and thus, we have to take care about the handling of them.
- **Listeners for Data Gain** We need to collect and save the metrics that are relevant for the systems and listeners are therefore a need.

- **Assertions** As an important part, we have to check if the test is successful and for this purpose we use assertions.

All the mentioned topics a crucial in terms of being successful with the tests and retrieving data that can be interpreted. We now explain the ideas behind the topics and explain how we designed the tests to support the above mentioned topics and their implementation.

6.1.1 User Load Controlling

An important topic is to control the load of users that is sending request to the targeted server. In our approach we have two types of tests. The first type provides a constant load of users over the complete test duration. The second type is a controlled user load that increases in a defined period of time. This type increases the load of users until a maximum user count is reached. JMeter and its plugins allow to configure both of the types and offers elements to add to the test plan, such as the "Ultimate Thread Group" as presented in figure 6.1.

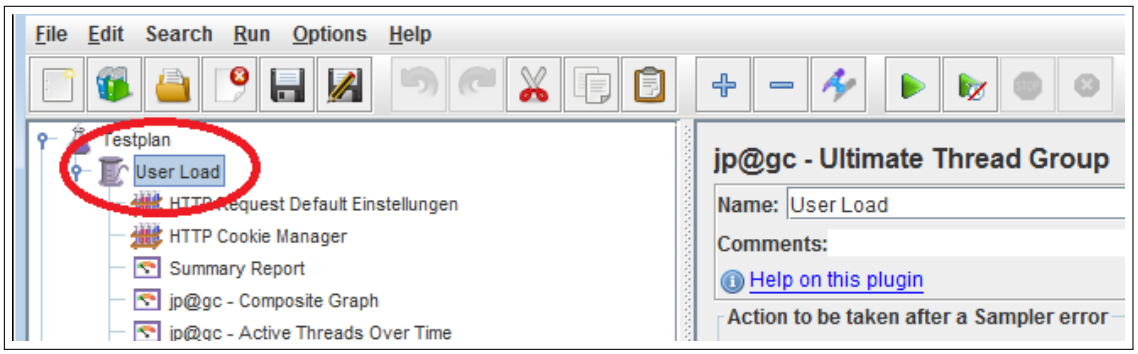


Figure 6.1: JMeter Ultimate Thread Group for Controlling the User Load

This element provides a possibility to configure a nearly arbitrary user load. In the configuration area of the element, the load can be defined and the tool presents the configured load in the form of a graph as presented in figure 6.2.

With this element we are able to configure all needed user loads and JMeter uses them for our tests to generate load for the target system.

6.1.2 Recording of Test Cases

To achieve a detailed and a true to original image of the tests and the requests that are sent during the test process, we use the recording feature of JMeter. It provides a so called "Test Script Recorder" that acts as a proxy server between the browser and the web application and records all requests for us. Subsequently the requests are listed and translated into JMeter elements to be able to replay the test. Figure 6.3 shows the recorded elements of a sample test.

For a better user readability, we rename the recorded HTTP(S) requests with the name of the pages that the request tries to load. Further, we check the requests if there are any obvious misrecordings.

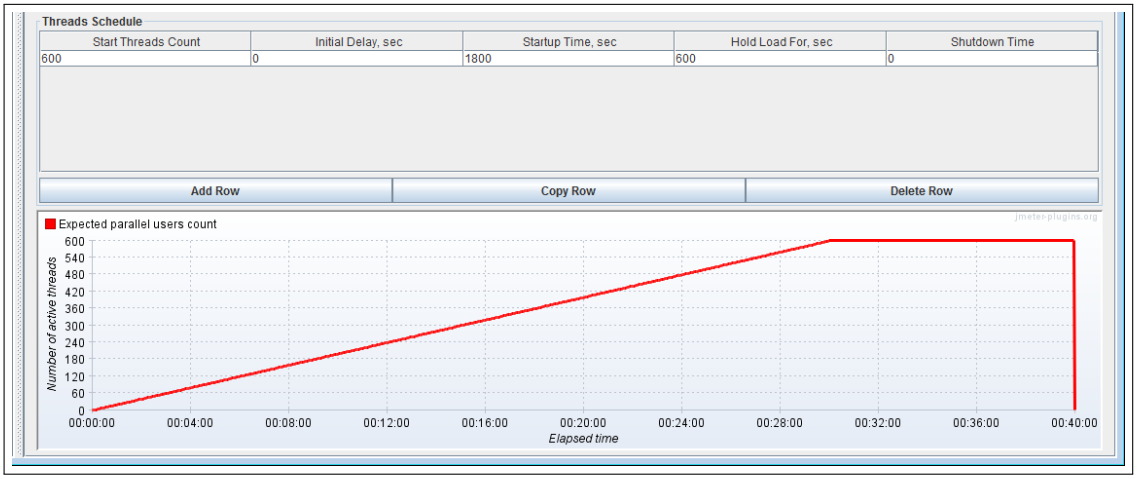


Figure 6.2: JMeter Ultimate Thread Group Configuration

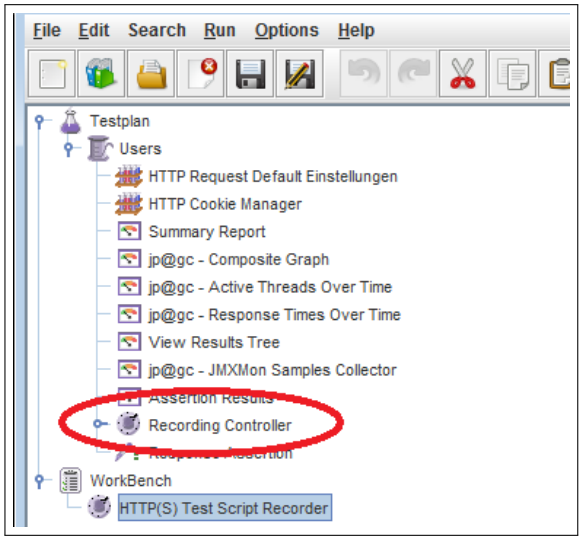


Figure 6.3: Elements recorded by the JMeter Proxy

6.1.3 Session Handling

As sessions are a very important feature in modern web applications, we need to take care of the session that might be important for further pages after a login. JMeter offers strategies for supporting this such as URL-rewriting or cookie management. As the target application uses cookies, we need to use the cookie management strategy for preserving the session. Figure 6.4 presents the element for that.

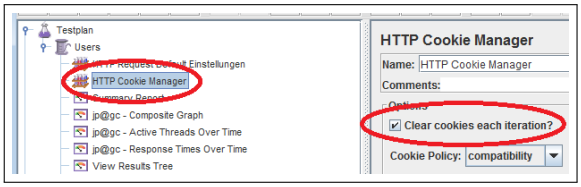


Figure 6.4: Cookie Managment in JMeter

We also need to ensure that each user of the target load gets its own session and therefore, we have to make sure that the session is cleared after each iteration of the test case. As seen in figure 6.4, the HTTP Cookie Manager offers a possibility to clear the cookie after each iteration.

6.1.4 Handling of Files and Paths

Another important feature is the proper handling of files as the use case needs a file upload of the signal data. Basically JMeter is able to detect POST requests and also multipart/form-data usage or page parameters. Figure 6.5 shows the parameters and file upload recognized by JMeter.

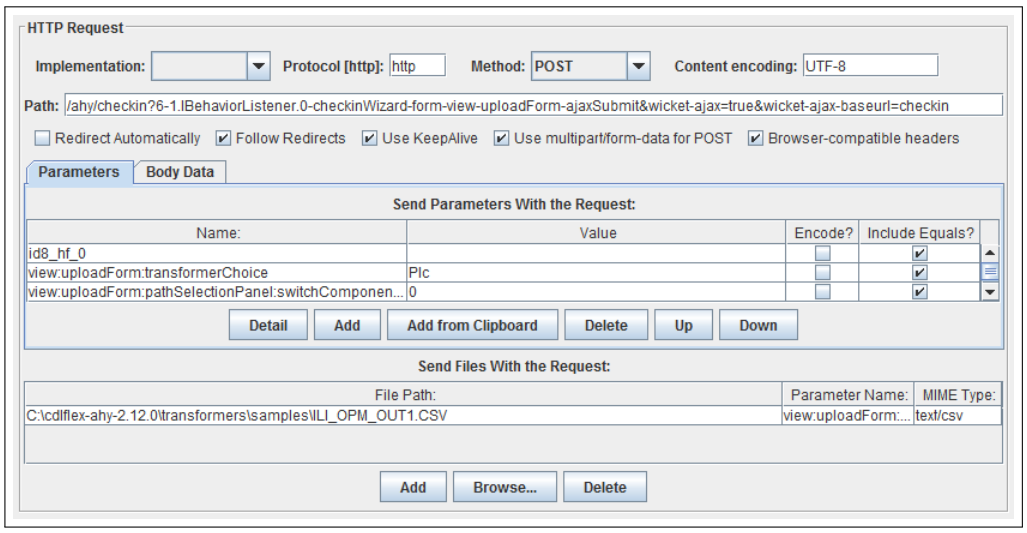


Figure 6.5: Handling of Files in JMeter

JMeter only records the relative file name, so consequently the path has to be adapted in order to have JMeter finding the correct file and being able to upload the file correctly.

6.1.5 Listeners for Data Gain

To make statements about a system's performance and stability, we have to record metrics during the test execution. As stated in section 4.2.3 we record the most important metrics to observe the system's performance. The elements of JMeter that record data are called listeners. Most of them - especially of the plugins - are also able to create graphs out of the box with the measured data as seen in figure 6.6. For our tests we mainly use the components "Summary Report", "Response Times over Time", "View Results Tree", "JMXMon Samples Collector" and "Assertion Results". The "Summary Report" and "Composite Graph" elements are used to sum up the data and to prepare reporting issues.

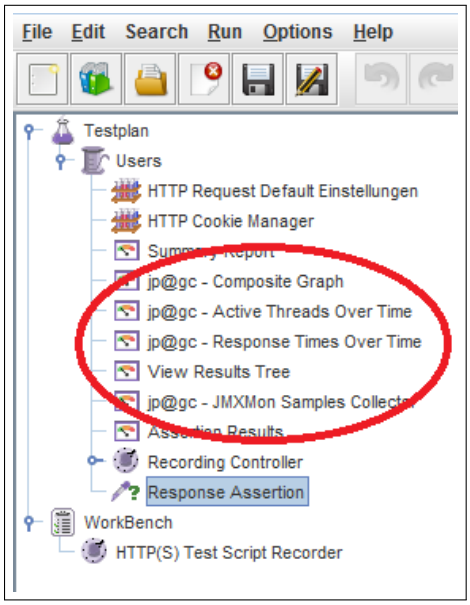


Figure 6.6: Listeners for Data recording in JMeter

Further, there are listeners for errors. As we have assertions for ensuring the application to run correctly even under heavy load, we also use these listeners to detect requests that basically return a page but not the expected one. These listeners are also able to write the results directly to a log file to post process the errors.

6.1.6 Assertions

While the test run, we want to ensure that a returned page is the page we expect, especially in some cases. We need to assert some data that we expect to be on the page and to exclude common error pages. For internal errors we add a global assertion such that if the error ever occurs during the test run, the assertion listener recognize that and raise an error.

For each request, such as the Checkin Page request or the Checkin Finish Request, we add assertion for the response page. We expect the respective page to be returned after the request.

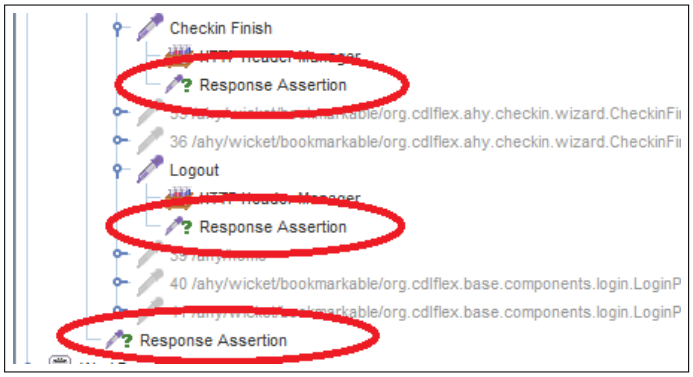


Figure 6.7: Response Assertions in JMeter

Therefore, we add response assertions for those pages and check the page for expected data. Figure 6.7 shows the assertions in the test case.

6.2 Checkin Page Tests

Our first type of tests is reaching the Checkin Page itself. We check if the page is able to handle a certain load. For that purpose The following steps are performed:

1. Call the Home Page
2. Login with the user data
3. Call Checkin Page

We test this procedure with different loads for 30 minutes each. The first run is with one user, the second run with 10 users and the last run is with 25 users. So from this type of tests we gain three tests to execute.

Figure 6.8 shows the complete test of the first three tests. Test 1 to 3 only differ in the amount of users that is configured to generate load. As we can see in this figure, we added HTTP Request Default Settings for setting the host and the port default values and a Cookie Manager for handling the session. Further, we added the above described listeners for collecting the data of the tests. Finally, the Recording Controller holds the HTTP requests that are needed to access the Checkin Page.

6.3 Checkin Process Tests

The second test type is the full Checkin Workflow. We check if the application can handle multiple Checkins in a row. The following steps are performed:

1. Call the Home Page

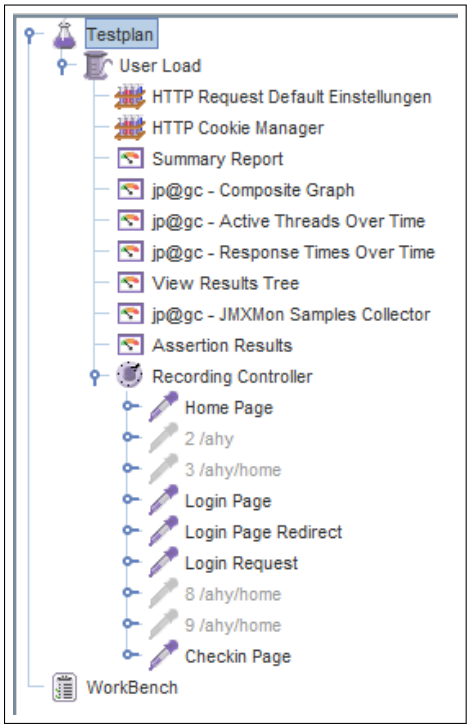


Figure 6.8: Complete Test Type 1

2. Login with the user data

3. Call Checkin Page

4. Upload Signal File

5. Proceed the wizard until finished

6. Logout

We test this procedure with different loads for 30 minutes each. The first run is with one user, the second run with 10 users and the last run is with 25 users. So from this type of tests we gain three tests to execute.

Figure 6.9 shows the additional steps that are necessary to perform the steps of a complete Checkin Workflow. The additional steps contain the upload of the signal file and finishing the wizard. Equally to the previous test type, we add listeners to gather the data for the interpretation and the assertions to ensure the pages are returned correctly. In this test, especially the steps of the file upload and the finishing step of the wizard are time consuming and need a special observation.

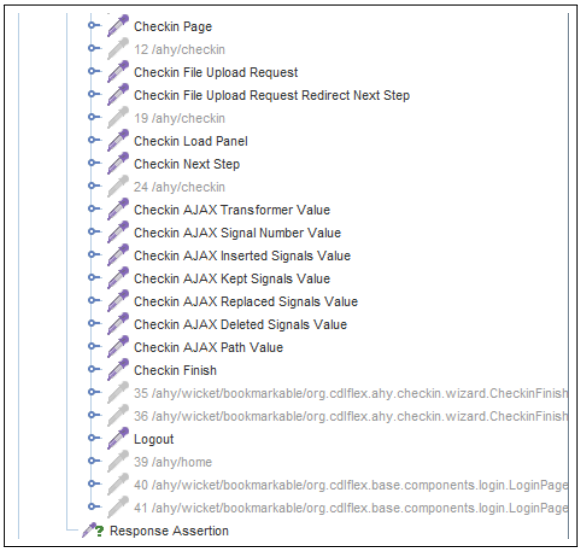


Figure 6.9: Additional steps for Test Type 2

6.4 Checkin Process Increasing Users Test

The last test type also implements the complete Checkin Workflow but differs from the other tests in terms of the user load and the test duration. We use the first 30 minutes of the test to steadily increase the amount of users that are requesting pages until the user counts reaches 600. The last 10 minutes of the test hold the load of 600 users. Figure 6.10 shows the expected parallel users count.

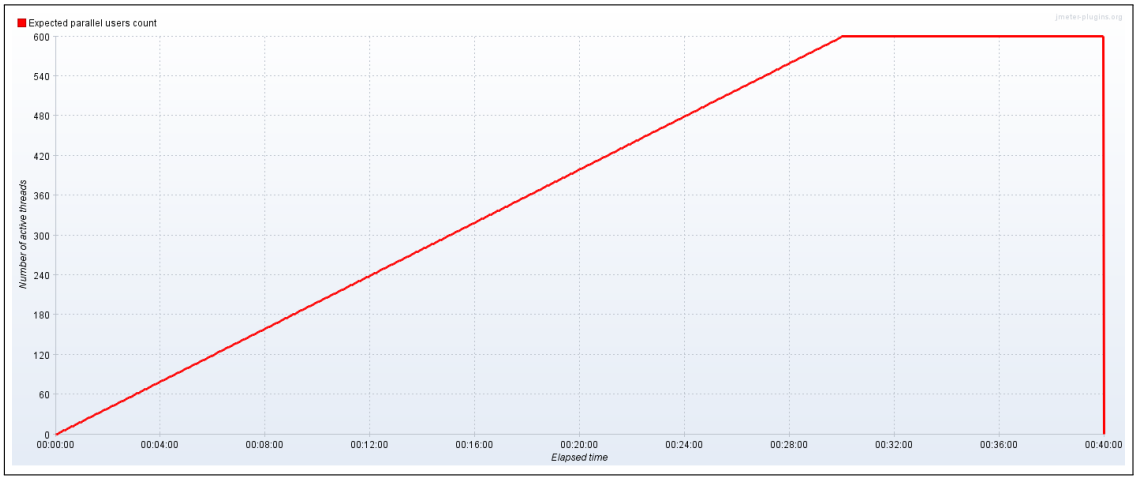


Figure 6.10: Planned User Load for Test Type 3

We want to check at what point of user load the server gets unable to serve the requests in a proper time and see if the systems breaks on any load. Also, we want to see if the system may

recover after the first occurrence of the breaking load.

In this chapter, we presented the results of the approach presented in section 4.2.3 to achieve a framework that is able to perform a performance analysis in the given research environment. As stated in section 3.3 the research issue 2.1 asks how a framework for performance tests can look like with the most promising tool - JMeter in our case. To answer this question we presented the design of the framework in this chapter as well as the implementation with JMeter which is asked in research issue 2.2.

CHAPTER 7

Test Framework Evaluation Results

In this chapter, we present the results of the case study that has been performed on the real world project of the web application front end. We present the results of the performance tests and compare them. We further state findings and argue for them. We present the results we encountered test type wise. The used computer has the configuration as stated in table 7.1

Component	Configuration
Processor	AMD FX-8350 8x4.00GHz
Memory	16GB DDR3
Operating System	Windows 7 64 bit

Table 7.1: PC Configuration

7.1 Reaching the Checkin Page - Type 1

This section presents the results of the first three tests of our planned tests of type 1. These results relate to the checkin page tests which try to login onto the web application and request the checkin page. We executed this test type with the load of 1, 10 and 25 users.

Figure 7.1 points out the trend of the response times of the 1 user test. We see the first period of time the application responds fast. After this period the response times especially for the Checkin Page increase heavily. A look into the logfiles revealed that this was most probably due to the start of garbage collection of the JVM. Obviously after the garbage collection the system recovers with lower response times but remains on a higher level than in the first period.

Further, the aggregated values of the test are calculated. Table 7.2 shows the values of the first test run with one user. As we can see, the average response time is good and even the maximum values are ok except of the Checkin page. As one would also expect, the error rate is 0% in this case.

1 User	MAX	MIN	AVG	MEDIAN	ERRORRATE
Checkin Page	7133 ms	551 ms	1034 ms	923 ms	0%
Home Page	922 ms	55 ms	83 ms	73 ms	0%
Login Page	870 ms	37 ms	54 ms	49 ms	0%
Login Page Redirect	866 ms	36 ms	53 ms	48 ms	0%
Login Request	1386 ms	80 ms	175 ms	105 ms	0%

Table 7.2: Results of Test Type 1 with 1 user

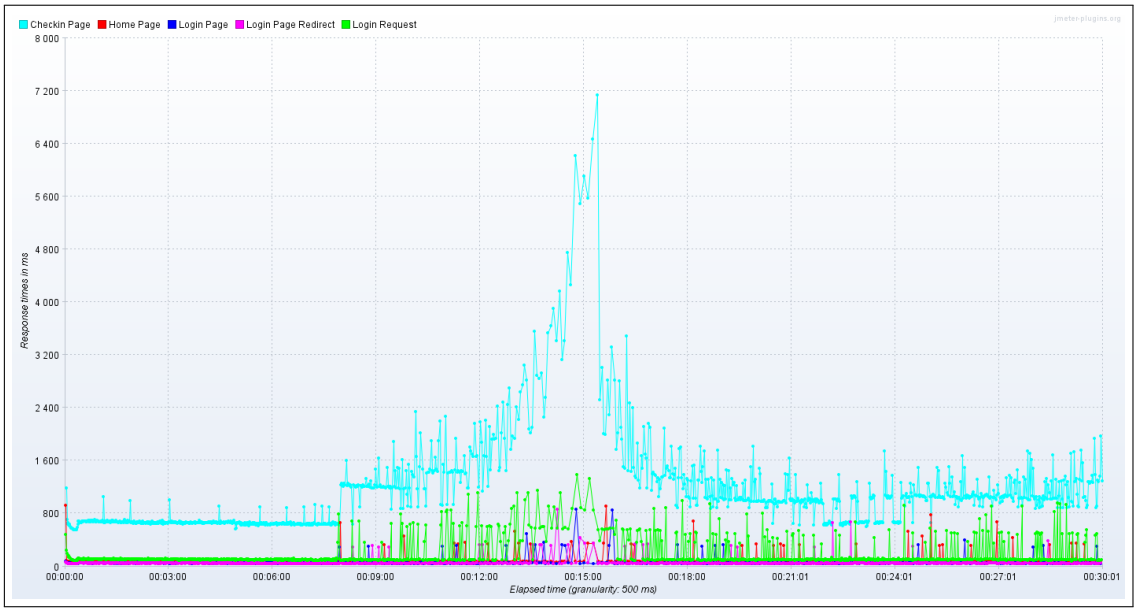


Figure 7.1: Response Times over Time - 1 user

As we also want to observe the server process' impact on the CPU load and the memory consumption, we measure the data of CPU load and heap usage. Table 7.3 shows the measured aggregated values of the CPU and heap usage. We can see that the average CPU load is in a good range and the maximum load is ok.

Figure 7.2 shows the full recordings of the memory consumption and CPU load. We can see that the the memory consumption increases over the time and even the maximum heap usage increases. We can further see a remarkable peak at about 15 minutes. At this point the garbage collection mechanism is invoked. Therefore, the CPU load increases and the memory consumption decreases. Equally to the response times, the used heap level then keeps on a stable level but a little higher than in the first period.

Concerning the heap usage of the next two tests with 10 parallel users and 25 parallel users, the behavior of the CPU load and memory consumption is very similar. In both the same garbage collection "outlier" can be seen and the increase and decrease of the maximum heap space is also there. Figures 7.5 and 7.6 present the actual trend during the whole test. Further, tables

Memory/CPU 1	cpuLoad	maxHeap	usedHeap
MAX	61%	1.017 MB	994 MB
MIN	0%	910 MB	154 MB
AVG	24%	968 MB	744 MB
MEDIAN	24%	974 MB	763 MB

Table 7.3: Results of Test Type 1 with 1 user

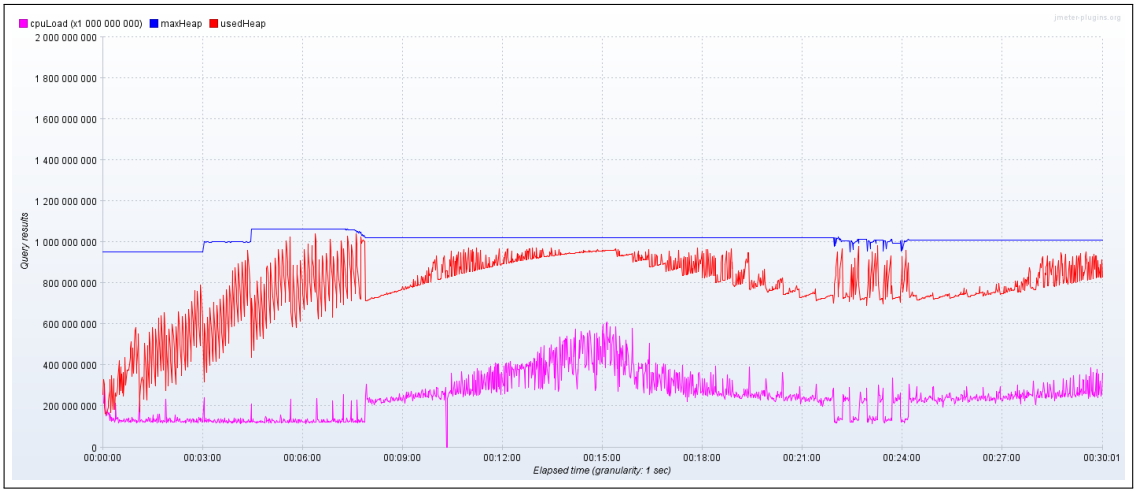


Figure 7.2: CPU and Memory Usage - 1 us

7.6 and 7.7 list the aggregated values of the user-enhanced type 1 tests. We can see a slightly higher maximum CPU usage and a doubled average CPU usage which is as expected due to the massive increased user load. Concerning the maximum heap space and the used heap space there is hardly any remarkable difference in all of the three tests of this type.

10 Users	MAX	MIN	AVG	MEDIAN	ERRORRATE
Checkin Page	350558 ms	1377 ms	10682 ms	5172 ms	0%
Home Page	70813 ms	71 ms	1073 ms	415 ms	0%
Login Page	42870 ms	43 ms	646 ms	179 ms	0%
Login Page Redirect	43663 ms	40 ms	627 ms	186 ms	0%
Login Request	180344 ms	274 ms	4071 ms	1570 ms	0%

Table 7.4: Results of Test Type 1 with 10 users

Further, the response times of the tests with 10 and 25 users can be seen in figures 7.3 and 7.4. We can see that the first dramatic raise of response times especially at the Checkin Page and the Login Request might also be a result of a lack of memory. The peak of about 600 seconds of response time may also be the reason for the non zero error rate in the last test. Some request probably simple time out if there is such a long waiting for the response.

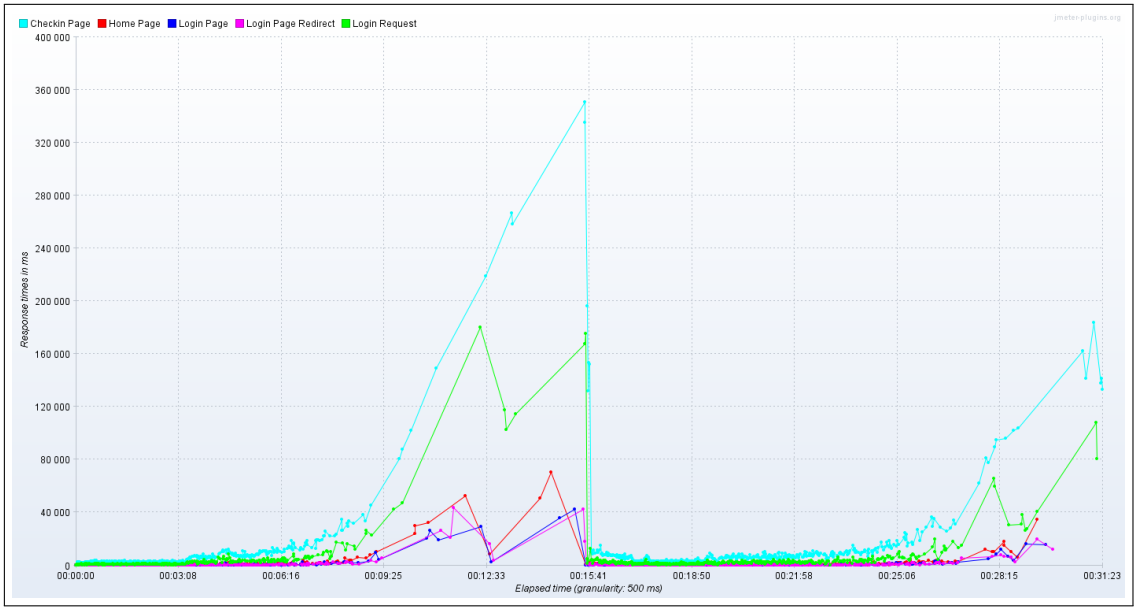


Figure 7.3: Response Times over Time - 10 users

25 Users	MAX	MIN	AVG	MEDIAN	ERRORRATE
Checkin Page	588409 ms	4112 ms	28003 ms	9613 ms	1%
Home Page	30152 ms	71 ms	1072 ms	369 ms	0%
Login Page	27847 ms	30 ms	573 ms	181 ms	0%
Login Page Redirect	89570 ms	29 ms	709 ms	176 ms	0%
Login Request	502602 ms	442 ms	11850 ms	2550 ms	0,16%

Table 7.5: Results of Test Type 1 with 25 users

Memory/CPU 10	cpuLoad	maxHeap	usedHeap
MAX	73%	1.014 MB	962 MB
MIN	19%	911 MB	147 MB
AVG	45%	953 MB	813 MB
MEDIAN	43%	960 MB	860 MB

Table 7.6: Results of Test Type 1 with 10 users

Memory/CPU 25	cpuLoad	maxHeap	usedHeap
MAX	74%	1.015 MB	988 MB
MIN	0%	911 MB	98 MB
AVG	44%	943 MB	799 MB
MEDIAN	42%	953 MB	846 MB

Table 7.7: Results of Test Type 1 with 25 users

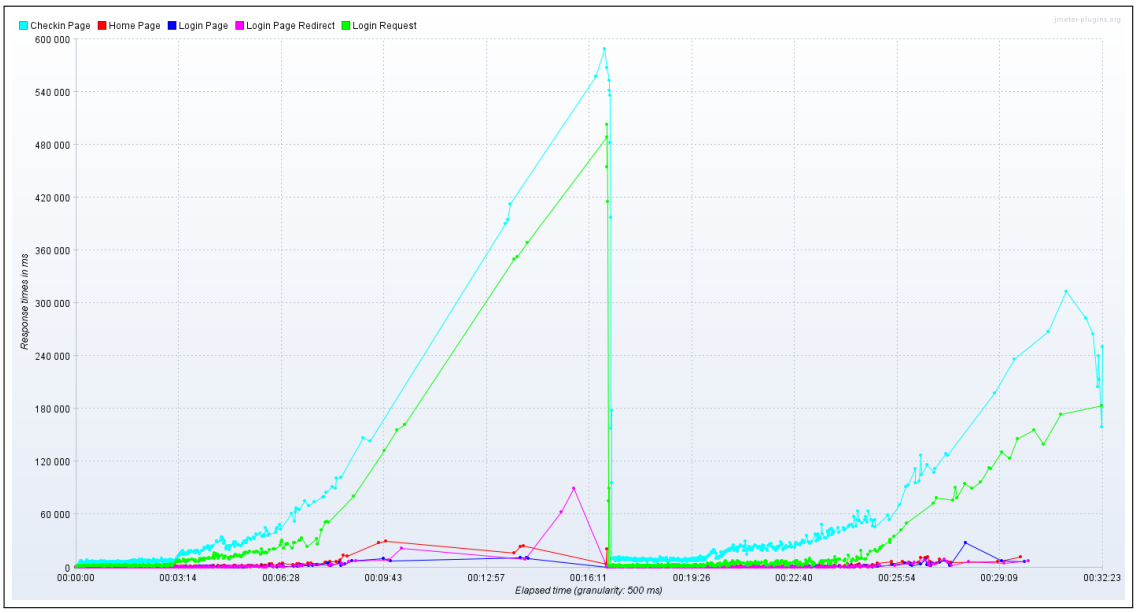


Figure 7.4: Response Times over Time - 25 users

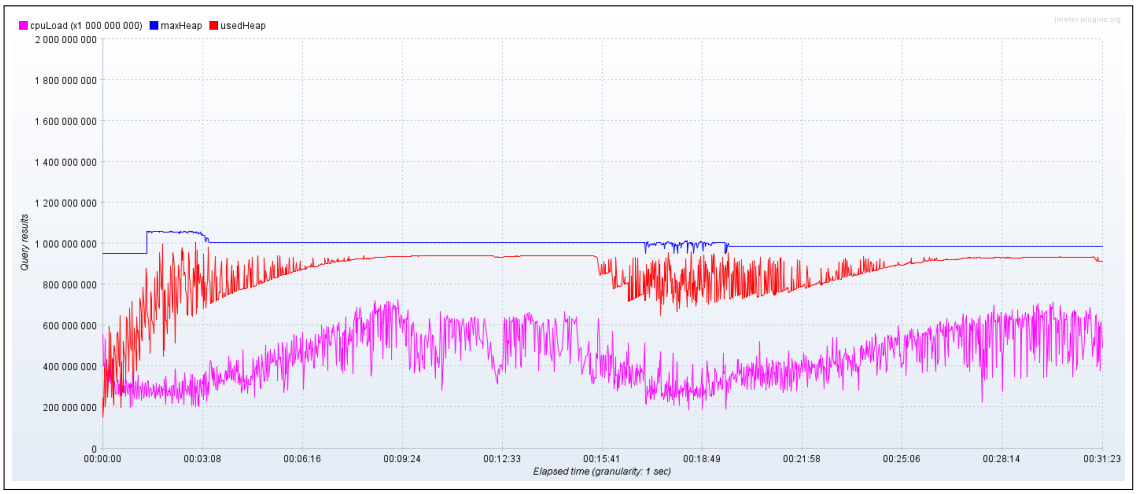


Figure 7.5: CPU and Memory Usage - 10 users

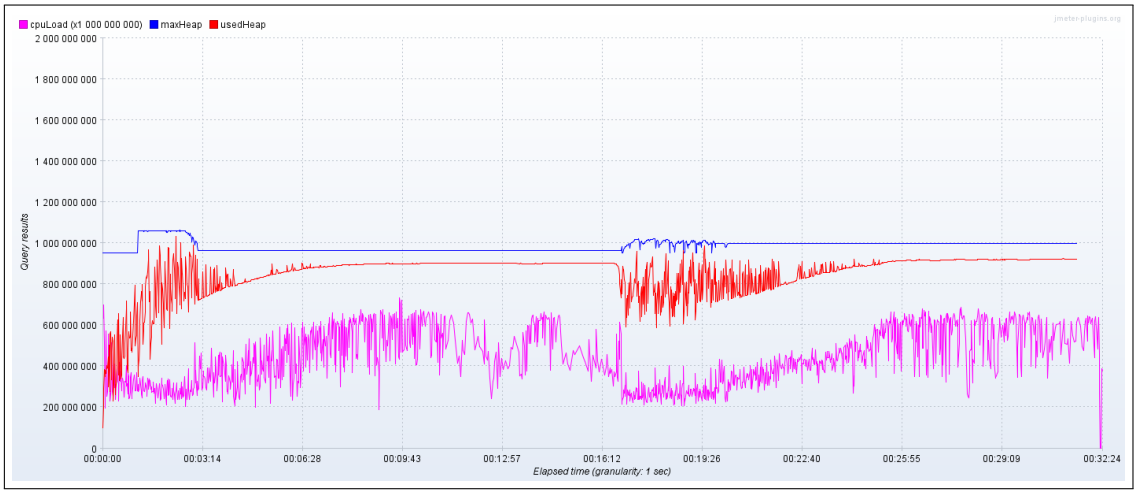


Figure 7.6: CPU and Memory Usage - 25 users

As a conclusion, we can definitely state that with an increasing number of users the server gets under stress and latest with 25 users there already occur errors in the responses. The two main critical pages in this type of test, the Checkin Page and the Login Request, massively increase the response time if load is added. Regarding our three tests we can illustrate the raise seen in table 7.8 also in figure 7.7.

Average Response Time	**1 User**	**10 Users**	**25 Users**
Checkin Page	1034 ms	10682 ms	28003 ms
Login Request	174 ms	4070 ms	11850 ms

Table 7.8: Average Response Time per User Load

7.2 Full Checkin - Type 2

This section presents the results of our planned tests of type 1. These results relate to the checkin page tests which try to login onto the web application and request the checkin page and further perform a file upload and a complete checkin process. We executed this test type with the load of 1, 10 and 25 users.

As seen in figure 7.8 the response times are essentially higher than in the previous tests especially for the new steps such as “Checkin Finish” and “Checkin Load Panel”. We can see a fast increase of the “Checkin Finish” response time at the beginning and at some point the response time drops to a much lower level and moves on with a slight pitch. The “Checkin Load Panel” increases the response time from the same point on. We can see that all this action is settled higher than in the previous tests.

We can also see this effect in the aggregated values table 7.9. The average value of the top pages is higher than 30 seconds and the maximum response bursts the level of 60 seconds. The

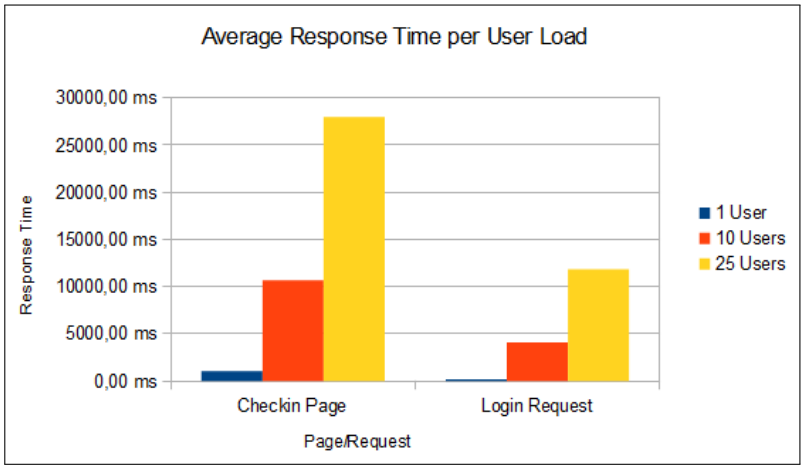


Figure 7.7: Average Response Time per User Load

overall performance is lower due to the higher amount of work that has to be done during the checkin process.

1 User	MAX	MIN	AVG	MEDIAN	ERRORRATE
Checkin Deleted	135 ms	113 ms	123 ms	123 ms	0%
Checkin Inserted	136 ms	113 ms	122 ms	123 ms	0%
Checkin Kept Signals	135 ms	111 ms	121 ms	121 ms	0%
Checkin Path	129 ms	102 ms	121 ms	121 ms	0%
Checkin Replaced Signals	133 ms	109 ms	121 ms	124 ms	0%
Checkin Signal Number	152 ms	105 ms	122 ms	122 ms	0%
Checkin Transformer	136 ms	105 ms	125 ms	125 ms	0%
Checkin Upload	4759 ms	2134 ms	2689 ms	2535 ms	0%
Checkin Upload Redirect	507 ms	381 ms	472 ms	477 ms	0%
Checkin Finish	167800 ms	5155 ms	40629 ms	14955 ms	0%
Checkin Load Panel	76690 ms	1227 ms	35549 ms	31844 ms	0%
Checkin Next Step	594 ms	467 ms	528 ms	527 ms	0%
Checkin Page	1189 ms	621 ms	734 ms	698 ms	0%
Home Page	174 ms	63 ms	76 ms	71 ms	0%
Login Page	62 ms	43 ms	50 ms	50 ms	0%
Login Page Redirect	76 ms	41 ms	50 ms	47 ms	0%
Login Request	461 ms	120 ms	165 ms	153 ms	0%
Logout	90 ms	70 ms	81 ms	81 ms	0%

Table 7.9: Results of Test Type 2 with 1 user

Surprisingly, the memory usage - as seen in figure 7.9 - keeps steadily and noticeable below the maximum heap value. Also the CPU usage is very constant and keeps on its level below

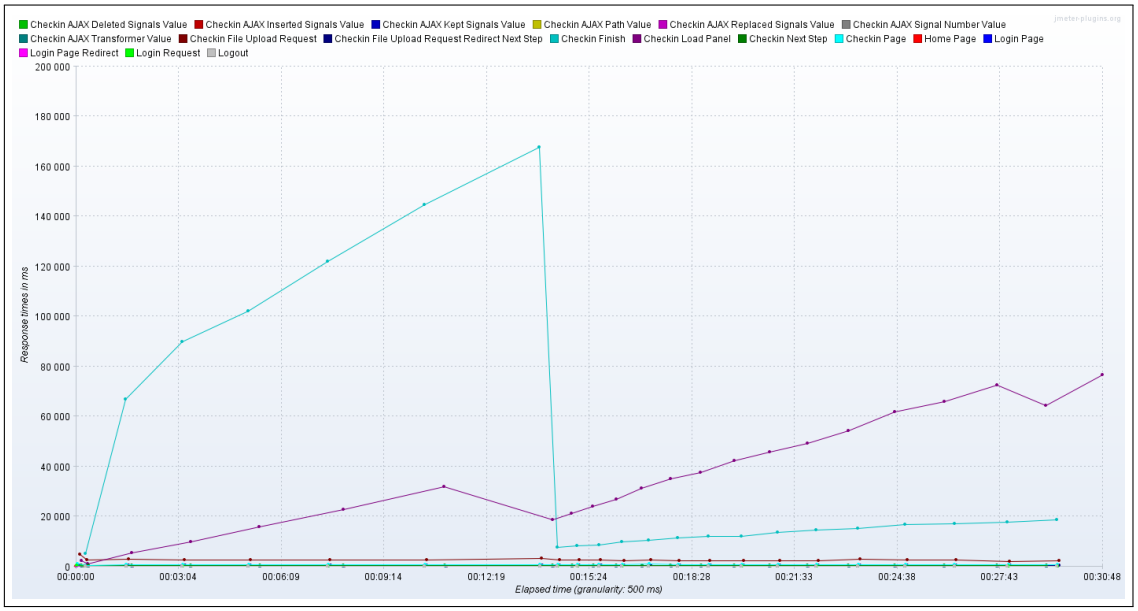


Figure 7.8: Response Times over Time - 1 user

20%. The values of table 7.10 also show the surprisingly low memory usage and CPU usage.

Memory/CPU 1	cpuLoad	maxHeap	usedHeap
MAX	38%	911 MB	629 MB
MIN	0%	911 MB	110 MB
AVG	13%	911 MB	339 MB
MEDIAN	13%	911 MB	336 MB

Table 7.10: Results of Test Type 2 with 1 user

Concerning the tests with 10 and 25 parallel users we expect an even higher impact on the systems performance. We can see in figure 7.10 and 7.11 that the response times increase a lot. It seems that there is a pattern of response times. The value increases to a certain value and than drops to a lower level. The “Checkin Finish Page” only comes with a few very high outliers. The aggregated values which can be seen in tables 7.11 and 7.12 show the same indication. In both test cases the “Checkin Finish Page” and the “Checkin Load Panel” have very high average response time.

Further, the memory consumption and CPU usage is inconspicuous as seen in figures 7.12 and 7.13. The memory usage increases to a level near the maximum but then drops and again starts to increase. The CPU usage is around 15% and steadily keeps on this level in both use cases. Tables 7.13 and 7.14 substantiate the indications seen in the figures.

To sum up the second type of we can conclude that while performing a complete checkin process the system reaches unusable regions of response times. Further, a very important finding

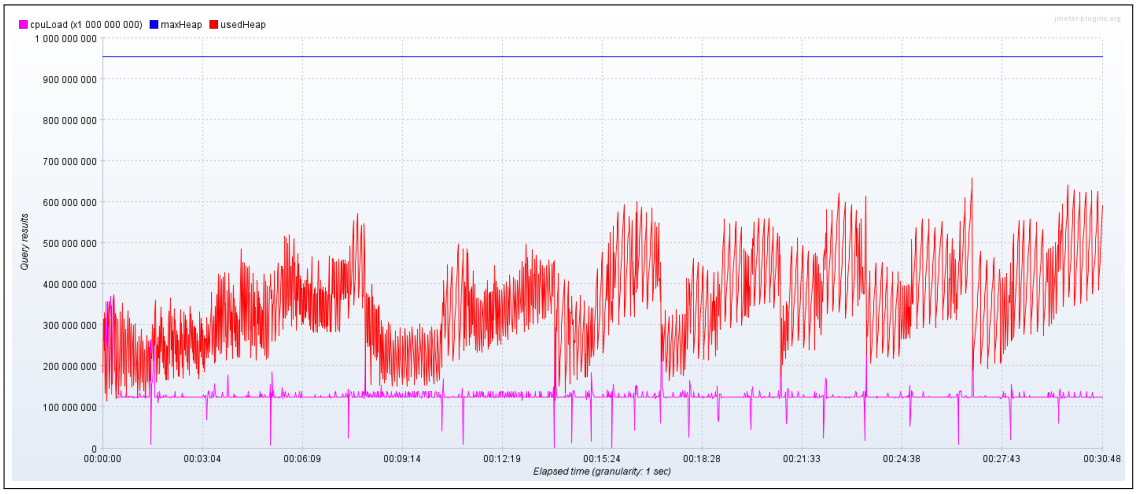


Figure 7.9: CPU and Memory Usage - 1 user

10 Users	MAX	MIN	AVG	MEDIAN	E-RATE
Checkin Deleted	48870 ms	4 ms	2453 ms	7 ms	0%
Checkin Inserted	39828 ms	4 ms	1330 ms	6 ms	0%
Checkin Kept Signals	30545 ms	5 ms	1586 ms	6 ms	0%
Checkin Path	49298 ms	4 ms	2607 ms	7 ms	0%
Checkin Replaced Signals	38394 ms	4 ms	2177 ms	6 ms	0%
Checkin Signal Number	99353 ms	4 ms	3045 ms	7 ms	0%
Checkin Transformer	51169 ms	199 ms	5791 ms	1232 ms	0%
Checkin Upload	50511 ms	562 ms	4592 ms	2274 ms	0%
Checkin Upload Redirect	102636 ms	507 ms	12530 ms	4325 ms	59%
Checkin Finish	1774215 ms	1144 ms	143311 ms	10516 ms	92%
Checkin Load Panel	97796 ms	886 ms	18353 ms	9491 ms	0%
Checkin Next Step	222022 ms	1090 ms	33756 ms	14839 ms	0%
Checkin Page	206640 ms	2410 ms	39072 ms	9448 ms	0%
Home Page	17245 ms	117 ms	4106 ms	371 ms	0%
Login Page	10084 ms	48 ms	1957 ms	213 ms	0%
Login Page Redirect	16510 ms	48 ms	2243 ms	198 ms	0%
Login Request	177380 ms	973 ms	30797 ms	3075 ms	0%
Logout	55301 ms	3 ms	12313 ms	1678 ms	2%

Table 7.11: Results of Test Type 2 with 10 users

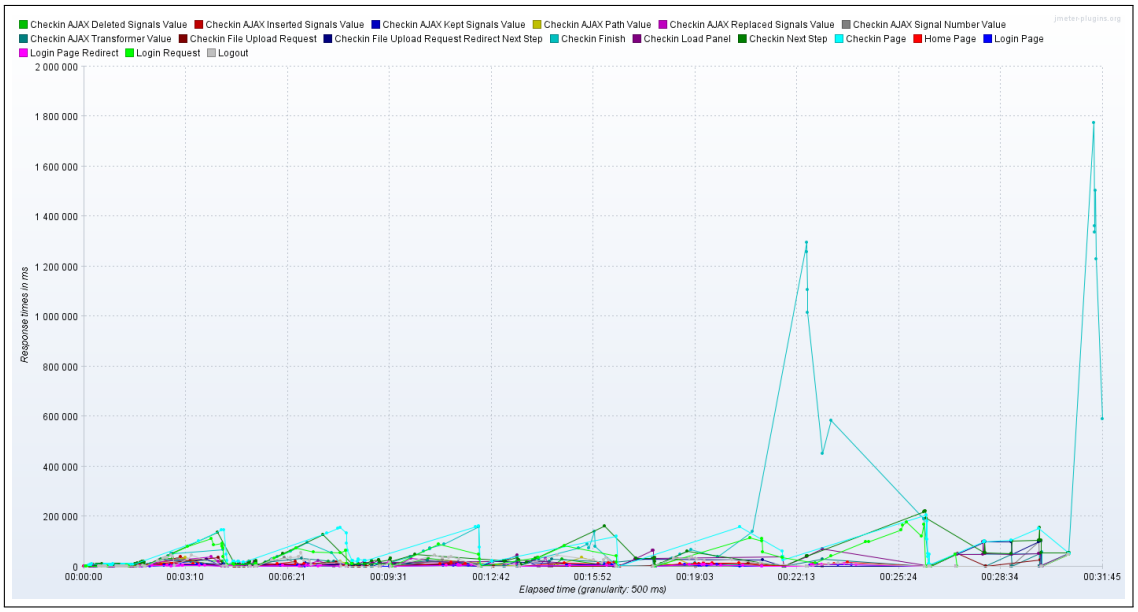


Figure 7.10: Response Times over Time - 10 users

25 Users	MAX	MIN	AVG	MEDIAN	E-RATE
Checkin Deleted	87911 ms	4 ms	4050 ms	6 ms	0%
Checkin Inserted	87606 ms	4 ms	2102 ms	6 ms	0%
Checkin Kept Signals	95289 ms	4 ms	5482 ms	6 ms	0%
Checkin Path	95900 ms	4 ms	2680 ms	6 ms	0%
Checkin Replaced Signals	97585 ms	4 ms	5578 ms	6 ms	0%
Checkin Signal Number	96639 ms	4 ms	3925 ms	6 ms	0%
Checkin Transformer	87842 ms	482 ms	8529 ms	1544 ms	0%
Checkin Upload	64731 ms	770 ms	10509 ms	2861 ms	0%
Checkin Upload Redirect	182907 ms	1727 ms	42701 ms	32271 ms	72%
Checkin Finish	1439471 ms	2272 ms	124613 ms	8826 ms	99%
Checkin Load Panel	198615 ms	3064 ms	64216 ms	55614 ms	0%
Checkin Next Step	205875 ms	2990 ms	61130 ms	46487 ms	0%
Checkin Page	310994 ms	4668 ms	56477 ms	26531 ms	0%
Home Page	65912 ms	148 ms	1240 ms	298 ms	0%
Login Page	93375 ms	50 ms	2106 ms	173 ms	0%
Login Page Redirect	74942 ms	78 ms	1231 ms	165 ms	0%
Login Request	103818 ms	532 ms	14586 ms	2844 ms	0%
Logout	66684 ms	2 ms	4635 ms	1403 ms	5%

Table 7.12: Results of Test Type 2 with 25 users

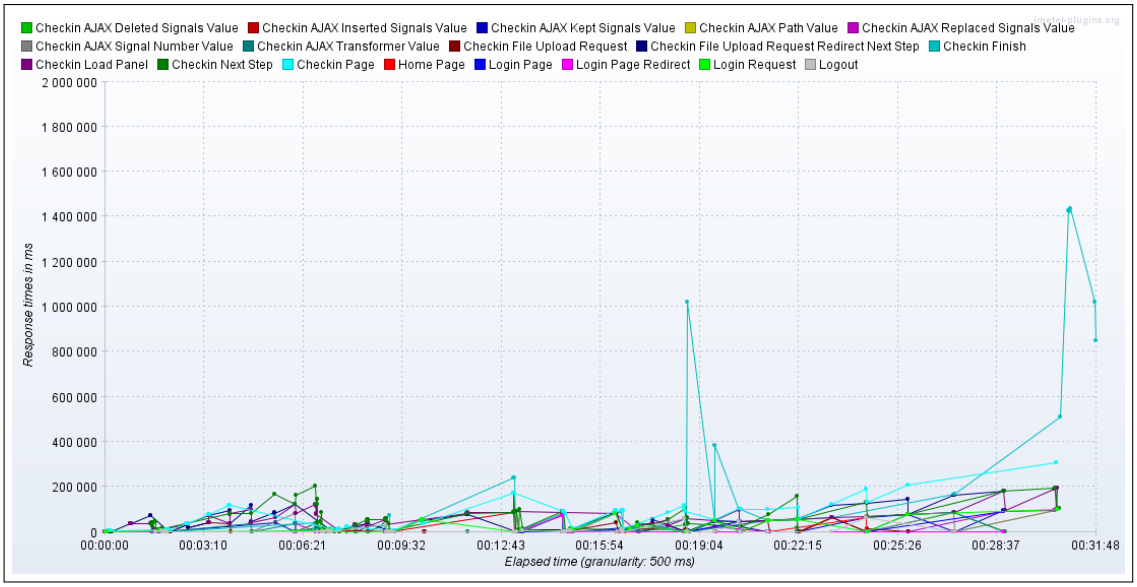


Figure 7.11: Response Times over Time - 25 users

Memory/CPU 10	cpuLoad	maxHeap	usedHeap
MAX	86%	1.020 MB	958 MB
MIN	0%	911 MB	96 MB
AVG	15%	992 MB	584 MB
MEDIAN	13%	999 MB	585 MB

Table 7.13: Results of Test Type 2 with 10 users

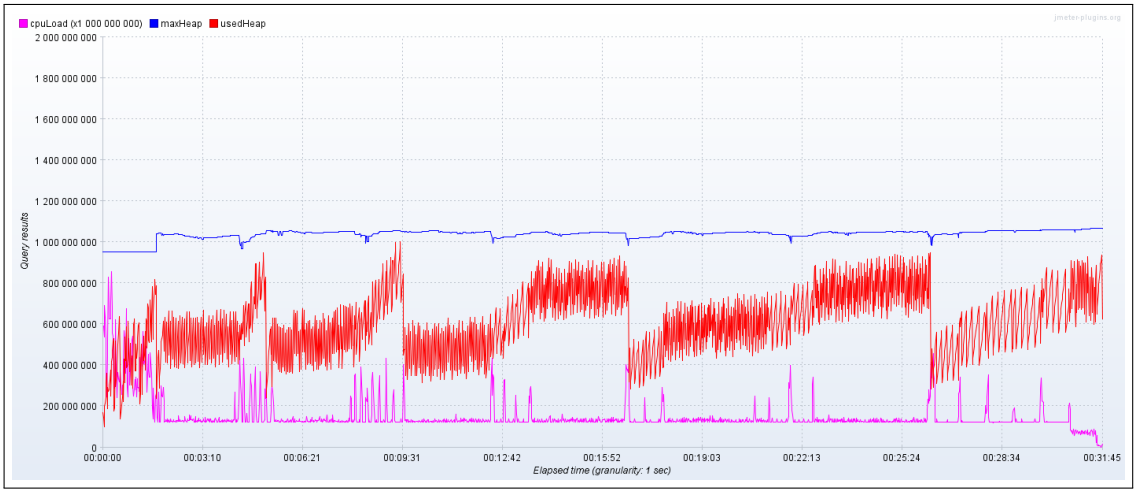


Figure 7.12: CPU and Memory Usage - 10 users

Memory/CPU 25	cpuLoad	maxHeap	usedHeap
MAX	53%	1.010 MB	979 MB
MIN	0%	911 MB	228 MB
AVG	14%	984 MB	592 MB
MEDIAN	13%	990 MB	592 MB

Table 7.14: Results of Test Type 2 with 25 users

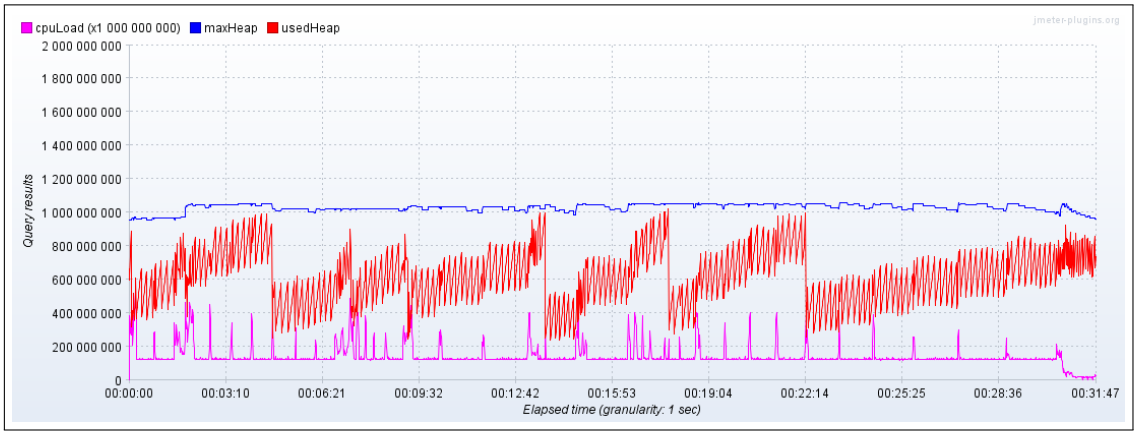


Figure 7.13: CPU and Memory Usage - 25 users

is that the error rate increases and makes it nearly impossible to reliably perform a checkin as more than 90% of the checkins fail. Figure 7.14 and table 7.15 illustrate the case:

Average Response Time	1 User	10 Users	25 Users
Login Request	165 ms	30797 ms	14586 ms
Checkin File Upload Request	2688 ms	4591 ms	10508 ms
Checkin Page	733 ms	39072 ms	56477 ms
Checkin Finish	40628 ms	143310 ms	124612 ms
Checkin Load Panel	35549 ms	18353 ms	64215 ms
Logout	81 ms	12313 ms	4635 ms

Table 7.15: Average Response Time per User Load

7.3 Maximum Load - Type 3

Our last test type is the increasing load test. Related to the previous results, we expect this test to result in a very low performance as the user count is steadily increasing. This expectation is approved by the tables 7.16 and 7.17 and the figures 7.15 and 7.16. The memory consumption stays at a high level of memory usage but is stable after a phase of increasing. The response

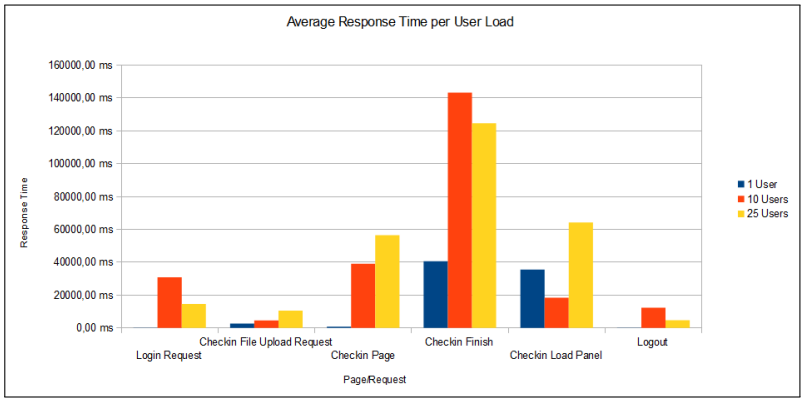


Figure 7.14: Average Response Time per User Load

times vary in a wide range but are all high after some users are added. At first the application answers fast but after the user count increases the response times get higher. At the end of this test, the response time is very inconstant.

Increasing Users	MAX	MIN	AVG	MEDIAN
Checkin Deleted	358001 ms	5 ms	37200 ms	1966 ms
Checkin Inserted	322929 ms	4 ms	36424 ms	3461 ms
Checkin Kept Signals	326593 ms	5 ms	43430 ms	3333 ms
Checkin Path Value	328988 ms	4 ms	35880 ms	1391 ms
Checkin Replaced Signals	321456 ms	4 ms	39577 ms	2408 ms
Checkin Signal Number	334259 ms	5 ms	44948 ms	13763 ms
Checkin Transformer	355813 ms	7 ms	117821 ms	109047 ms
Checkin Upload	591504 ms	591 ms	126646 ms	76909 ms
Checkin Upload Redirect	1254005 ms	219 ms	253730 ms	177460 ms
Checkin Finish	2637983 ms	412 ms	369478 ms	225998 ms
Checkin Load Panel	534459 ms	372 ms	149263 ms	100143 ms
Checkin Next Step	404009125 ms	714 ms	2137189 ms	368858 ms
Checkin Page	1387670 ms	644 ms	402603 ms	289855 ms
Home Page	551616 ms	82 ms	143705 ms	36391 ms
Login Page	358431 ms	34 ms	98387 ms	29999 ms
Login Page Redirect	248470 ms	31 ms	68371 ms	27631 ms
Login Request	453726 ms	210 ms	98852 ms	72725 ms
Logout	604979 ms	241 ms	153856 ms	121185 ms

Table 7.16: Results of Test Type 3 with increasing users

The research issue defined in section 3.4 asks if the presented approach is feasible and if it works properly in the research environment. We have shown that the our framework with the most promising tool is able to test a web application of the research environment for performance

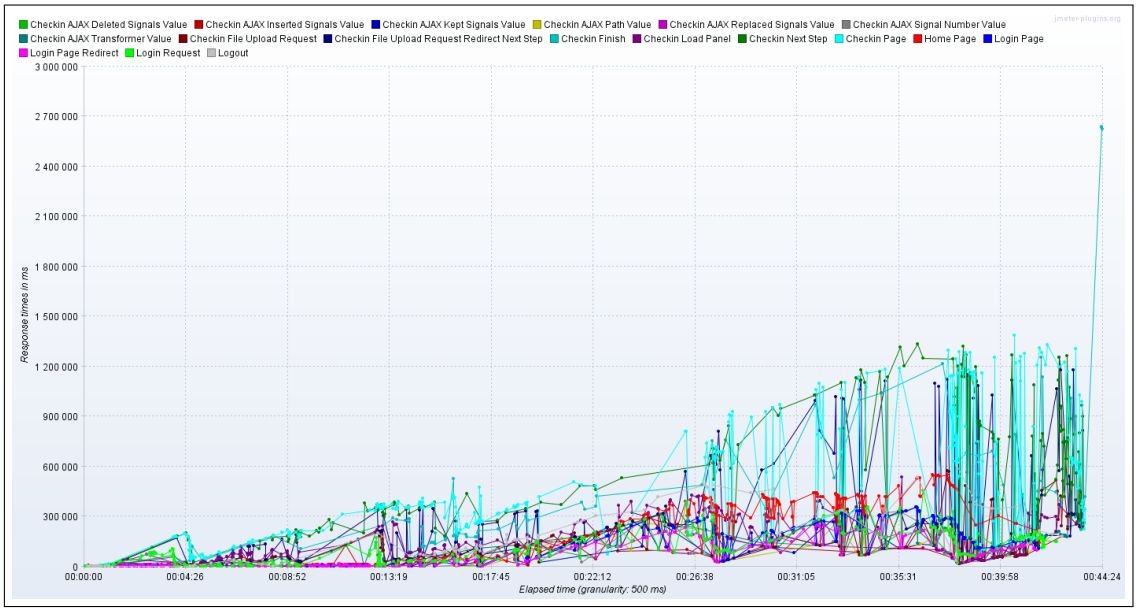


Figure 7.15: Response Times over Time - Increasing users

issues. We have shown that the framework can be applied and returns feasible results as in this case study.

Memory/CPU 1	cpuLoad	maxHeap	usedHeap
MAX	74%	1.013 MB	984 MB
MIN	0%	911 MB	92 MB
AVG	25%	954 MB	734 MB
MEDIAN	23%	953 MB	767 MB

Table 7.17: Results of Test Type 3 with increasing users

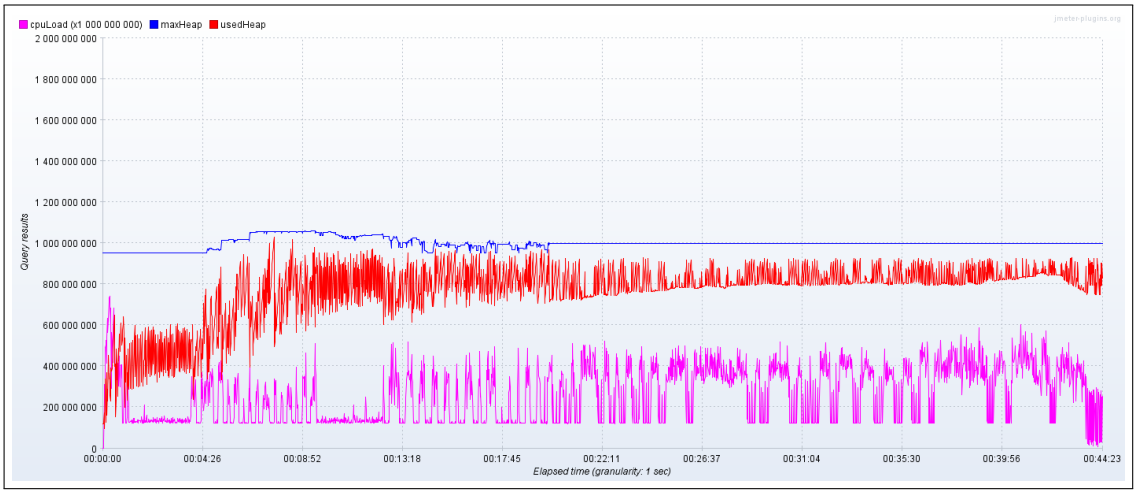


Figure 7.16: CPU and Memory Usage - Increasing users

CHAPTER 8

Discussion and Limitations

In this chapter we critically examine and discuss the study and show up possible limitations.

8.1 Discussion

This section discusses the results that are achieved in chapters 5, 6 and 7, discusses the connection to the related work of chapter 2 and shows the feasibility of the study and the fulfillment regarding the research questions.

In the related work we presented various approaches of performance testing and an approach to evaluate tools. We use this scientific context for our study. In the tool selection part of this work, we use the approach of Robert M. Poston and Michael P. Sexton . On the one hand we determine a list of criteria which represents the user needs of the research environment. Further, we use the second part of the evaluation study after Robert M. Poston and Michael P. Sexton to evaluate a best fitting tool based on the criteria list. With this result, we can proceed with designing a test framework for the research environment. We use a prototyping approach to design this framework. The last part uses a case study approach to apply the framework and to see its feasibility. We now discuss each part of the thesis in detail.

In this thesis, the overall target is to perform performance analysis in a specialized area. The area this research is located in, is the area of heterogeneous distributed systems in an automation systems environment. We describe this area in chapter 3. For this thesis we concentrate on the change management workflow that is used in the research prototype and reflects one of the most crucial and complex features of the system under test. This workflow is depicted in figure 3.2. To be able to perform a performance analysis on this prototype and in this research area we can derive three main research questions that arise out of this.

- **Tool Selection** This research question deals with the selection of a tool with we want to perform the analysis and splits up in three research issues we identified. We present them in detail in the sections below.

- **Test Framework Design and Implementation** In this research question, the main target is to create a test framework based on the given environment. We identified two parts at this point. We go into the details in the sections below.

- **Test Framework Evaluation** As a consequence of creating a framework for performance analysis, we need to ensure that the proposed framework is feasible in terms of our application and to verify its usability. A more detailed view on this research question is provided below.

These three main research questions are identified due to the main goal of performing performance analysis in the research environment. Their location in the research environment around the ASB is shown in figure 8.1

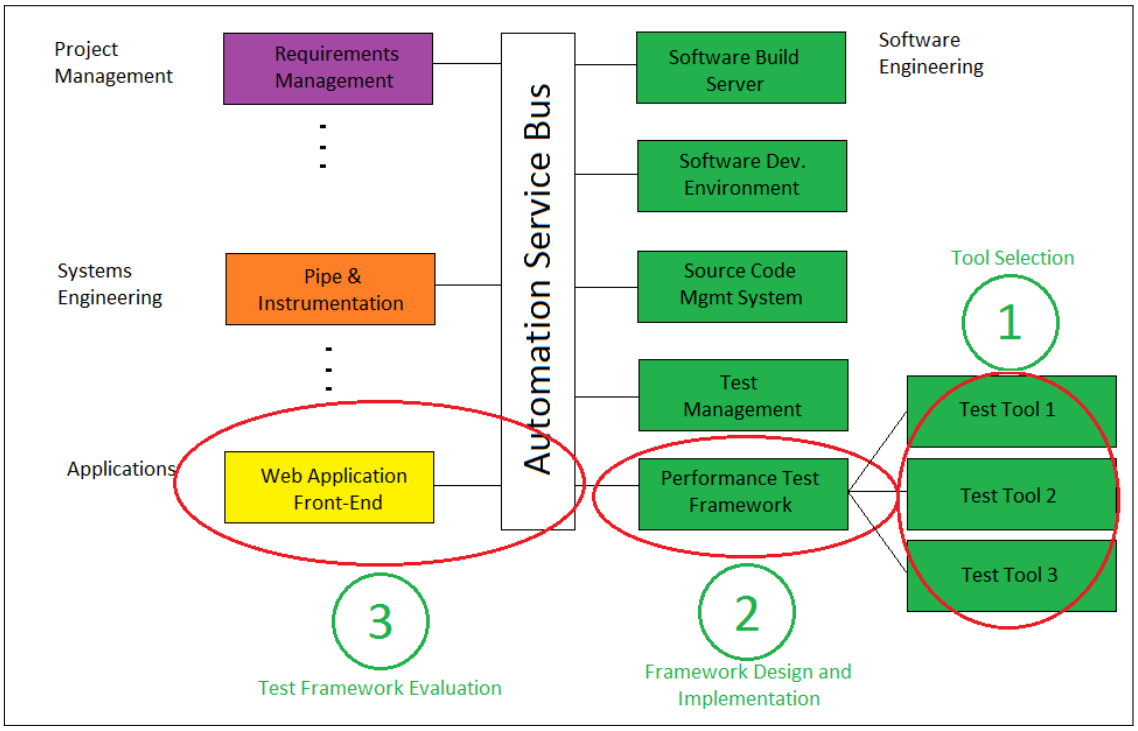


Figure 8.1: Illustration of the location of the research problems in the (extracted) environment

We can see that RQ1 - Tool Selection - is located outside and is used for evaluating a most suitable tool (refers to number 1 in figure 8.1. One of the tools will be selected by an approach after Robert M. Poston and Michael P. Sexton [74] and is the tool of choice for this work. We describe our approach in detail in the sections below. RQ2 - Test Framework Design and Implementation - uses the chosen tool and creates an approach for a framework that is able to guide and control the performance analysis process in our environment. The so created test framework also contains high level test cases to specify important aspects to test and cares about general issues of performance testing in web applications too. RQ3 - Test Framework Evaluation

- uses the implemented approach of performance testing that is a contribution of RQ2 and aims to check the framework for feasibility and applicability. Therefore, the framework is applied on the research prototype of the web front-end and checked as it is stated in figure 8.1. We now revisit each research question in detail and present the connected research issues.

8.1.1 Research Question 1 - Tool Selection

We now take a deeper look in to research question 1, the tool selection. In the previous sections we already describe the main sketch of this question. As the overall target of this work is to perform performance analysis, there is a need for a tool that is able to support all requirements that are needed in our research environment. Hence, we need a list of requirements that reflect those needs and we have to choose a tool according to this needs. To implement this evaluation in a strategic and scientific way, we adapt the approach of Robert M. Poston and Michael P. Sexton [74] which is a guideline for tool evaluation studies. The overall approach looks like illustrated in figure 8.2:

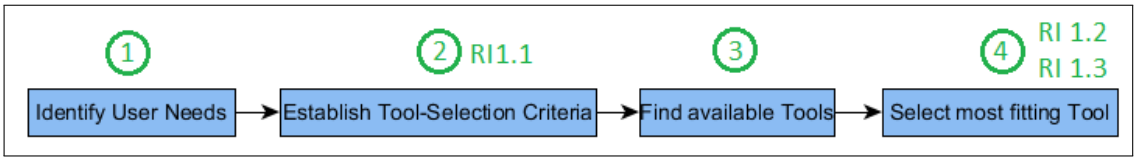


Figure 8.2: Main Steps of the Approach of Poston and Sexton [74]

In research question 1 we asked for a tool that is able to perform performance analysis in our research environment. According to the research issues mentioned in chapter 3, we can answer the research issues as follows:

- **RI 1.1:** *What are the requirements for performance testing in the research environment?* We found a list of criteria that reflect the requirements for performance testing in the research environment. Those requirements are grouped in four main categories and can be seen in figure 5.1 and figure 5.2 for a detailed view.

  First we identify the user needs and collect the requirements that are necessary. In this study, this step is done by an environment analysis within the lab. This is labeled with number 1. The next step is to extract criteria for the tool selection out of the user needs which reflects number 2 in figure 8.2. Further, all available tools have to be found (number 3) and the most fitting tool is selected (number 4) after rating the tools. We use this approach and define weights for the criteria as seen in table 4.1. These weights refer to the criteria that are found in chapter 5. The criteria are listed in figure 5.1 and figure 5.2 and are the answer of research issue 1.1. Based on these results, we found a list of criteria that represents the requirements of the users and thus can be used to find a tool in our context.

- **RI 1.2:** *How do the available tools fulfill the requested requirements?* Based on this list of criteria we were able to rate the tools to get an overall impression of the tool in terms of our criteria. The detailed results can be seen figures in 5.12, 5.13 and 5.14.

With this approach and the full list of available tools, which can be seen in Appendix A, we can now evaluate the tools by rating the tools according to the criteria. The first part is to create a short list of tools by evaluating the mandatory criteria. The resulting short list we received is illustrated in table 8.1.

As we can see in table 8.1 we end up with a list of 10 tools at the shortlist (the full list contains 95 tools). According to the proposed approach, these tools are evaluated with the optional criteria. The full evaluation results can be seen in figures 5.12, 5.13 and 5.14 which is also the answer of research issue 1.2. Due to the usage of the proposed approach by Robert M. Poston and Michael P. Sexton [74], this evaluation is efficient and effective. A summary of the results is listed in table 8.1. Based on this result, we can evaluate tools with the criteria with the tools and achieve a rating for each tool.

- **RI 1.3:** *Which of the available tools or set of tools is the best for the environment?* The full list of scored tools with which was presented in RI1.2 leads to JMeter as the most suitable tool for our purposes. Table 8.1 shows the overall scores and the selection of JMeter.

ID	Tool Name	Achieved Points (rounded)	Achieved Score (rounded)
37	JMeter	133	95%
74	SilkPerformer	121	86%
9	CLIF	112	80%
91	WAPT	108	77%
54	NeoLoad	105	75%
24	Grinder	104	74%
83	Testing Anywhere	101	72%
48	LoadComplete	99	70%
11	Contiperf	55	39%
85	TestStudio	-	-

Table 8.1: Final Result of the Evaluation - Ranking of the Tools

In this table the tools and the relating achieved points and achieved score are presented. As we can see JMeter is the most suitable tool for our purposes. It received 133 points and a total score of 95%. This answers research issue 1.3. Therefore, we use JMeter for our implementation of the framework.

8.1.2 Research Question 2 - Test Framework Design and Implementation

Based on the results of the previous study - the Tool Selection - we can proceed with thinking about a procedure that is able to control and guide the performance analysis process in the research environment. There are already frameworks that deal with testing in this context such es from Winkler et al. [86] but not in the context of performance testing. Hence, we have to adapt the framework for our purposes of the environment. We split up this research question in two separate parts and research issues.

- **RI 2.1:** *How does a framework for performing performance tests look like with the most promising tool and in the research environment?* The first one deals with the design of the framework and relates to research issue 2.1 presented in chapter 3. We therefore design a framework based on the approach of Winkler et al. [86]. Our approach is illustrated in figure 8.3.

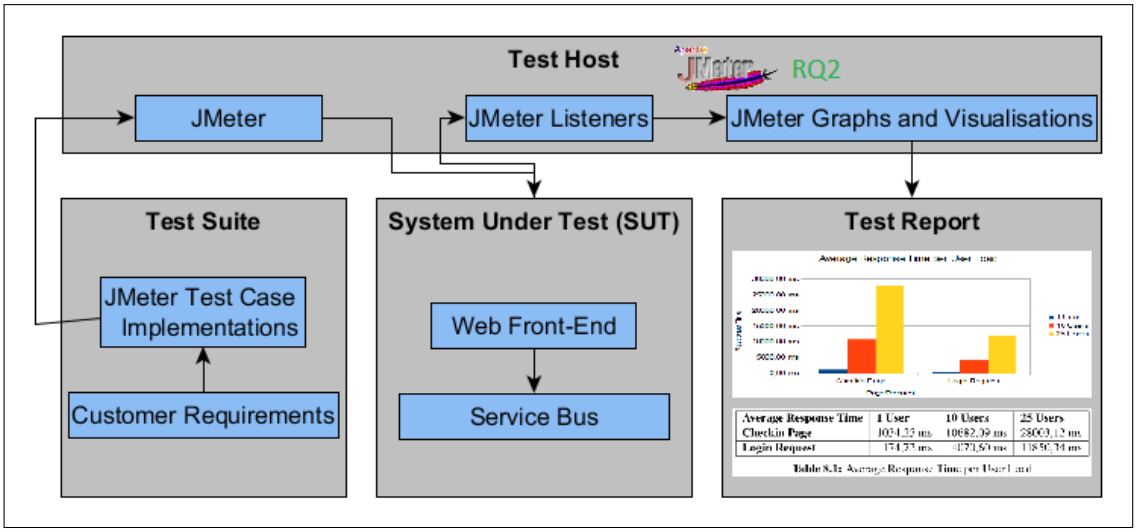


Figure 8.3: Test Framework Implementation

We define a Test Suite in figure 8.3 that consists of the customer requirements and JMeter Test Case Implementations derived from the customer requirements. We describe these test cases below in detail. The next component is the Test Host. It contains the Test Framework as well as the JMeter Listeners (which collects the data gained by the tests) and the JMeter Graphs and Visualisations. The last component is the Test Report. This framework is introduced to control and guide the performance analysis in this thesis and further answers research issue 2.1.

Further, we define test cases which have to be run on the target. We define three types of tests that have again three tests defined. They are:

1. **Type 1** - Perform the Checkin Process only until Step 1 is displayed (figure 4.5)
   a) Request Page before critical steps with one user for 30 minutes
   b) Request Page before critical steps with 10 users for 30 minutes
   c) Request Page before critical steps with 25 users for 30 minutes
2. **Type 2** - Perform the Checkin Process completely as stated in figure 4.4 until the wizard is finished (figure 4.7)
   a) Perform Checkin Process with one user for 30 minutes
   b) Perform Checkin Process with 10 users for 30 minutes
   c) Perform Checkin Process with 25 users for 30 minutes the service breaks

3. **Type 3** - Perform the Checkin Process completely as stated in figure 4.4 until the wizard is finished (figure 4.7)
    a) Perform Checkin Process with a steadily increasing amount of users until the service breaks

Based on these tests, we achieve a full framework that is able to test our purposes and return results of the investigated system under test.

- **RI 2.2:** *How can the designed framework be implemented in the research environment?* We implemented this approach as seen in chapter 6. The resulting framework was implemented with the most suitable tool - JMeter. To that end, we defined general issues that might impair the framework implementation. These are mainly due to fact that the web front-end is a web application and has to be treated carefully. The general issues we discuss in chapter 6 are User Load Controlling, Recording of Test Cases, Session Handling, Handling of Files and Paths, Listeners for Data Gain and Assertions. We indicate potential problems and present solutions for the problems. Further, the proposed test cases are implemented with JMeter as figure 8.4 shows.

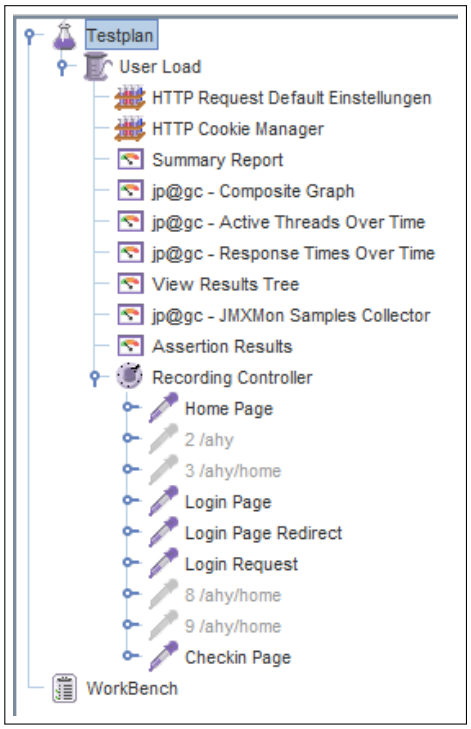


Figure 8.4: Complete Test Type 1 - Implemented with the test framework and the JMeter tool

Figure 8.4 shows an implemented test case of the test type 1, implemented with JMeter. We can see user load generator, requests of pages, assertions as well as data collectors such as graphs and data collectors. This answers research question 2.2.

8.1.3 Research Question 3 - Test Framework Evaluation

We have chosen a most suitable tool, designed and implemented a framework for performance testing in our research environment so far. As we want to ensure the quality of the research, we need to show that the framework is feasible and applicable. To that extend we can achieve two main contributions on this research question. First, we perform a feasibility study to show the framework is working in our environment and second, we perform a pilot evaluation to get an impression of the system under test (the web front-end and the underlying bus system) in terms of performance analysis. Hence, we can give a basic feedback to the developers which components might need a performance improvement. This is packed in research issue 3.1.

- **RI 3.1:** *Is the implemented approach feasible and does it work properly in the research environment?* We evaluated the framework on the web front-end and received various data which is presented in chapter 7 in detail. The main findings are presented here. The evaluation of the type 1 tests led to the following aggregated results:

Average Response Time	**1 User**	**10 Users**	**25 Users**
Checkin Page	1034 ms	10682 ms	28003 ms
Login Request	174 ms	4070 ms	11850 ms

Table 8.2: Average Response Time per User Load

Table 8.2 presents the aggregated results of the type 1 tests with a different amount of users. As this table reveals, the higher the amount of users generating load is, the longer the requests take to be finished. Further, we can see that the difference between two tests (for instance between 1 and 10 users or between 10 and 25 users) increase as well so that there might be an exponential increase of the response time depending on the user amount.

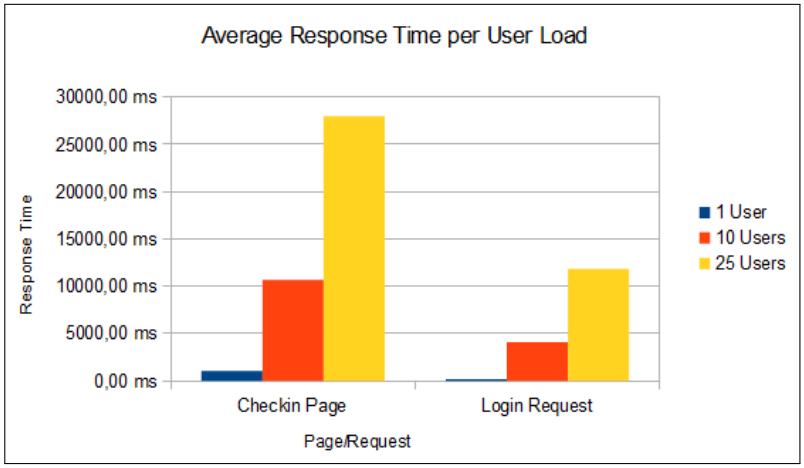


Figure 8.5: Average Response Time per User Load

Figure 8.5 emphasizes this finding in a graphical way. We can see that at both requests the response time increases. The type 2 tests show a similar behavior. Table 8.3 reflects

the response times of the various requests. We can see a remarkable increase of average response time already at a repeated checkin with a 1 user load. This suggests that there might be a performance issue at the Checkin Load Panel or the Checkin Finish step. The increasing response time on increasing user load is basically given but not for all steps.

Average Response Time	1 User	10 Users	25 Users
Login Request	165 ms	30797 ms	14586 ms
Checkin File Upload Request	2688 ms	4591 ms	10508 ms
Checkin Page	733 ms	39072 ms	56477 ms
Checkin Finish	40628 ms	143310 ms	124612 ms
Checkin Load Panel	35549 ms	18353 ms	64215 ms
Logout	81 ms	12313 ms	4635 ms

Table 8.3: Average Response Time per User Load

The fact that there is no strictly monotonically increasing average response time in all of the requests is better illustrated in figure 8.6. We can see that at Checkin Finished or Login Request there is a unpredictable behavior. This may be due to the fact that the application was already on the limit with 10 users.

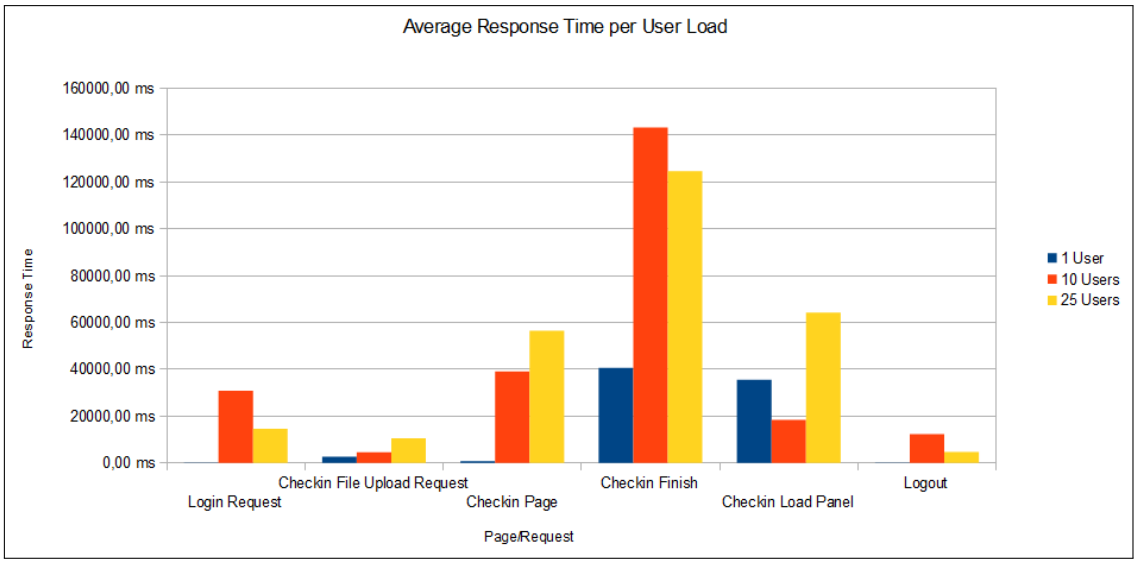


Figure 8.6: Average Response Time per User Load

For the type 3 tests, we encountered an early unpredictable behavior. We presented the results of the feasibility study and the performance analysis of the system under test. The values which our proposed framework collected and reported are applicable in terms of the experienced performance of the application. This is the answer of research issue 3.1. Based on these results we can conclude that the tool is feasible for our purpose and can be applied for performance testing in our environment.

We have now discussed the results based on the research questions and linked some possible limitations. We now explain the limitations and threats to validity of the study in detail in the next section.

8.2 Limitations and Threats to Validity

Based on the results and the proposed approach some limitations yield. In this section we identify and describe them. Furthermore, we discuss the threats to validity of this thesis.

8.2.1 Threats to Internal Validity

Internal validity deals with the content of the study itself and describes internal factors that might threat the study's validity. In this study we identified the following internal threats.

- **Criteria Selection** Due to the fact that the criteria selection is done by experts and users of the research environment, the selection is subjective and might not reflect the objective criteria for the selection. The selection may be driven by personal preference. Furthermore, the chosen criteria might not reflect the optimal criteria set. We address this threat as the selection is spread over many experts.

- **Completeness of Requirements** As we have considered a list of requirements to achieve the evaluation, it may be the case that some aspects have not been covered and may therefore missing in the criteria selection. We discussed the requirements with experienced industry experts of software and requirements engineering and therefore this threat is minimized.

- **Weightings and Ratings** Another task that is executed by a human is the mapping of criteria with weights. This may also be biased by personal preference. The same explanation is valid for the ratings that are done during the evaluation. Especially in case of Usability category. We counteract this thread by having experts for this evaluation.

- **Completeness of Tool List** As it is impossible to cover all available tools that exist to include them into the study, there might be a tool that has not been included in our study and therefore has not been evaluated. There may also be tools developed for personal usage that are not included in this list. This threat is minimized by a comprehensive search and a investigation of the most common tool sites.

- **Appearance of new Tools** The list of tools was created at the beginning of the study. As the study was executed over a period of time it is possible that new tools or new versions of listed tools have been released but have not been taken into account in this study. We minimize this threat by the method of evaluation. A new tool can easily be added to the evaluation and be included in it.

- **Inappropriate Framework Design** It could turn out that the framework design is not appropriate for the usage in this study or at all because it is tailored to the needs of the

current research stated in chapter 3. The framework design is an important part of this thesis and therefore this is a threat. We encounter this threat by using a well established framework [86] and adapt it for our purpose.

- **Representativeness of Test Set** Our test set is tailored for the use case that is described in chapter 3. Hence, the results of this particular area of the application under test may not be representative for the rest of the application. The other areas may present a better or a worse performance than the presented area.

8.2.2 Threats to External Validity

External validity deals with the ability to generalize the findings of the study. In this section we present the threats to external validity. The thesis is developed in the context of the CDL-Laboratory and especially tailored for that. Due to this fact, the generalization of the chosen criteria and the evaluation as well as the implementation and the case study might not be working in other environments. In contrast to that, the evaluation process is a well established process [74] and has already been applied in different previous studies such as [34].

CHAPTER 9

Conclusion and Future Work

This chapter summarizes the work of this thesis, gives a conclusion and points out possible future work. We go again into the methodology and present a summary of the achieved results.

9.1 Goal of the Thesis

Software quality assurance is a crucial part of modern software development and as a part of it performance testing and analysis is as well. Even newer areas of Software Engineering have to implement a software quality assurance strategy. This is also valid for the area of heterogeneous distributed engineering environments. Software Engineering methods are applied on this field and so are software quality methods. In this thesis, we concentrate on performance testing in the field of heterogeneous environments.

In order to be able to peform performance testing, the first goal of this thesis is to choose a tool that supports our purpose. This is accomplished by having a scientific evaluation with a well recognized and approved approach by Robert M. Poston and Michael P. Sexton [74]For that, a long list of possible tools is created with the help of a research for available tools. Based on this long list, a short list is received by applying the mandatory criteria. Therefore, a list of criteria is determined and weights are mapped to the found criteria. Further, the criteria are used for evaluation to elect a most promising tool for the further study.

After the most promising tool is found the next goal of this thesis is to design a testing framework for our research environment that is able to test an application in our research area in terms of performance. The presented framework is then implemented with the most promising tool from the tool study above. The outcome of this goal is a performance test framework that is able to test the web application prototype of the research environment.

The last goal is to show that the designed and implemented framework is feasible for our purposes. We therefore perform a case study on the web application front end and apply the testing framework. We then examine the results of the tests in terms of sanity and feasibility and get an impression of the performance of the target application.

9.2 Methodology

We use different approaches for our three main thesis parts. The first part that deals with the tool evaluation is related to an approach of Robert M. Poston and Michael P. Sexton [74]. We adopt this approach if necessary. According to this approach we find criteria for the evaluation that reflect the user needs. These criteria are weighted to express the importance of each of the criteria items. Based on the criteria, we evaluate a short list of tools, which was created before by evaluating the must have criteria, according to the criteria list. The scores of each tool are then compared and the tool with the highest score is the most fitting tool for our purposes. The second part of the thesis creates a design for a testing framework for performance issues. The presented design is then implemented with the most promising tool from the tool evaluation. The last part uses a case study approach to determine the feasibility and sanity of our framework.

9.3 Tool Selection

We executed the evaluation for the criteria which ended up in a list of criteria with weights that can be seen in 5.2. The evaluation of the tools is described in chapter 5 in detail. We started with a list of 95 tools. After the must have evaluation, which can be seen in table 5.1, 10 tools are remaining. With this short list of tools we performed the detailed tool evaluation. The results of this evaluation can be seen in section 5.2.11. The evaluation resulted in JMeter being the most suitable tool of this study. The detailed results are listed in table 9.1

ID	Tool Name	Achieved Points (rounded)	Achieved Score (rounded)
37	JMeter	133	95%
74	SilkPerformer	121	86%
9	CLIF	112	80%
91	WAPT	108	77%
54	NeoLoad	105	75%
24	Grinder	104	74%
83	Testing Anywhere	101	72%
48	LoadComplete	99	70%
11	Contiperf	55	39%
85	TestStudio	-	-

Table 9.1: Final Result of the Evaluation - Ranking of the Tools

9.4 Test Framework Design and Implementation

The design of the framework discusses major aspects of using the tool in the research domain and with the application under test. Section 6.1 deals with user load controlling, session handling or assertions in terms of JMeter and the application under test. The following sections the describe each of the three test types in detail. The implementation details as well as possible pitfalls are presented.

9.5 Test Framework Evaluation

In the test framework evaluation, we determine if the proposed framework is feasibly for our research environment. Therefore the framework is applied on the web application and performance analysis is performed on it. As proposed in the framework design we have three types of tests. For all three tests we run the single tests and evaluate the data gathered hereby. We can see a high increase of the response times if the user count is increased. Figure 9.1 and 9.2 show the average response times of the pages. The detailed results can be seen in chapter 7.

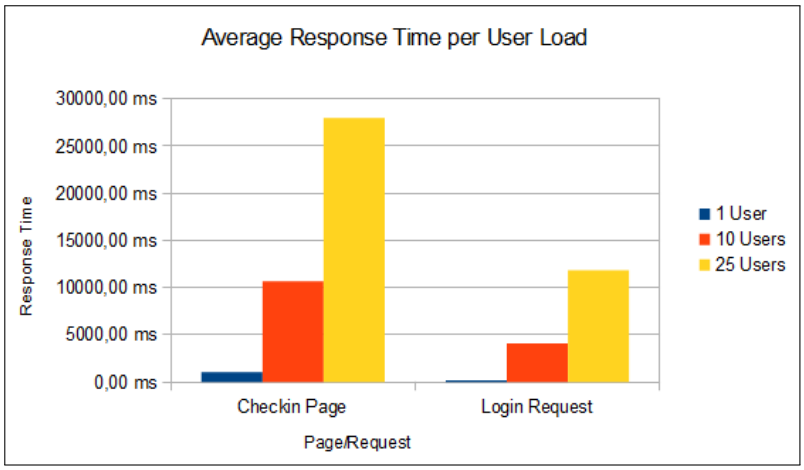


Figure 9.1: Average Response Time per User Load

9.6 Future Work

During the thesis we recognized some points that could lead to future work. While searching for tools we encountered many service hosts that offer performance tests in a cloud. As this could be a cheaper and more scalable solution for commercial tools it might be interesting for further research to examine the offers and to test cloud performance tools. In our thesis the tests were executed on one machine to simulate the user load. For further study it might be interesting to distribute the load generating clients on many different machines to achieve a more realistic load and a more distributed test. For our purposes of comparing the results and verifying the frameworks sanity it is no problem that the web application was run on a local server as we do

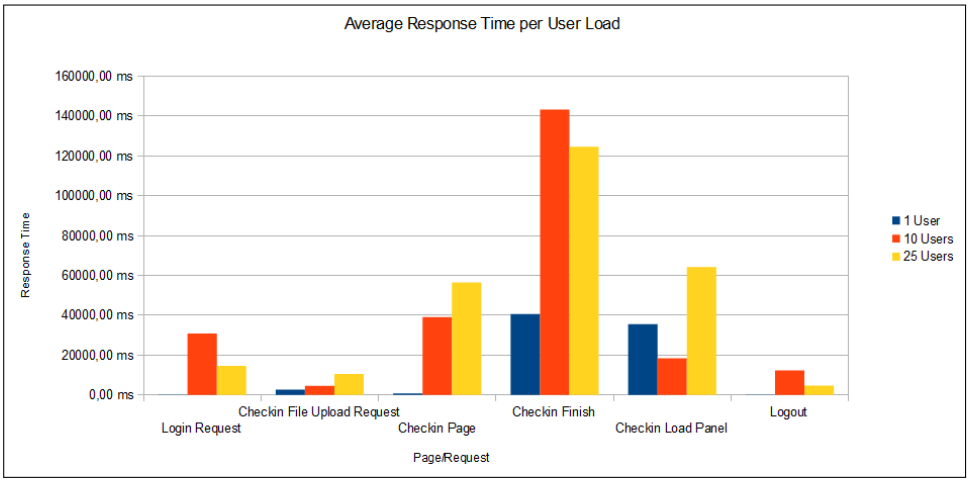


Figure 9.2: Average Response Time per User Load

not care about absolute values. For a detailed statement of the performance of the application, the tests must be repeated on a productive-similar environment. Furthermore, the test context should be extended. In other words there have to be more tests of other areas of the application to give a better statement about the application's performance. Another future task is to observe the market for new tools and new versions of already evaluated tools and repeat the evaluation if new interesting tools arise.

Bibliography

[1] Ieee trial-use standard reference model for computing system tool interconnections. *IEEE Std 1175*, pages 1–, 1992.

[2] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services*. Springer, 2004.

[3] LB Arief and Neil A. Speirs. A uml tool for an automatic generation of simulation programs. In *Proceedings of the 2nd international workshop on Software and performance*, pages 71–76. ACM, 2000.

[4] Martin Arlitt, Diwakar Krishnamurthy, and Jerry Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology*, 1(1):44–69, 2001.

[5] Alberto Avritzer, Andre Bondi, Michael Grottke, Kishor S. Trivedi, and Elaine J. Weyuker. Performance assurance via software rejuvenation: monitoring, statistics and algorithms. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 435–444. IEEE, 2006.

[6] Alberto Avritzer, Joe Kondek, Danielle Liu, and Elaine J. Weyuker. Software performance testing based on workload characterization. In *Proceedings of the 3rd international workshop on Software and performance*, pages 17–24. ACM, 2002.

[7] Alberto Avritzer and Brian Larson. Load testing software using deterministic state testing. In *ACM SIGSOFT Software Engineering Notes*, volume 18, pages 82–88. ACM, 1993.

[8] Alberto Avritzer and Elaine J. Weyuker. Generating test suites for software load testing. In *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pages 44–57. ACM, 1994.

[9] Alberto Avritzer and Elaine J. Weyuker. The automatic generation of load test suites and the assessment of the resulting software. *IEEE Transactions on Software Engineering*, 21(9):705–716, 1995.

[10] Alberto Avritzer and Elaine J. Weyuker. Deriving workloads for performance testing. *Software: Practice and Experience*, 26(6):613–633, 1996.

[11] Alberto Avritzer and Elaine J. Weyuker. Monitoring smoothly degrading systems for increased dependability. *Empirical Software Engineering*, 2(1):59–77, 1997.

[12] Alberto Avritzer and Elaine J. Weyuker. The role of modeling in the performance testing of e-commerce applications. *IEEE Transactions on Software Engineering*, 30(12):1072–1083, Dec 2004.

[13] Simonetta Balsamo, Marco Bernardo, and Marta Simeoni. Combining stochastic process algebras and queueing networks for software architecture analysis. In *Proceedings of the 3rd international workshop on Software and performance*, pages 190–202. ACM, 2002.

[14] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[15] Scott Barber. Beyond performance testing, parts 1–14. *IBM developer works, rational technical library*, 2004.

[16] Scott Barber. Creating effective load models for performance testing with incomplete empirical data. In *Proceedings of the 26th Annual International Telecommunications Energy Conference*, pages 51–59. IEEE, 2004.

[17] Simona Bernardi, Susanna Donatelli, and José Merseguer. From uml sequence diagrams and statecharts to analysable petri net models. In *Proceedings of the 3rd international workshop on Software and performance*, pages 35–45. ACM, 2002.

[18] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. Æmpa: a process algebraic description language for the performance analysis of software architectures. In *Workshop on Software and Performance*, pages 1–11, 2000.

[19] Stefan Biffl, Alexander Schatten, and Alois Zoitl. Integration of heterogeneous engineering environments for the automation systems lifecycle. In *Proceedings of the 7th IEEE International Conference on Industrial Informatics*, pages 576–581. IEEE, 2009.

[20] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, 1984.

[21] Barry W. Boehm, John R. Brown, and Hans Kaspar. Characteristics of software quality. 1978.

[22] Barbara M. Bouldin. *Agents of change: Managing the introduction of automated tools*. Pearson Education, 1988.

[23] special bonus issue Business Week. The qualitative imperative, 1991.

[24] David Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[25] Lawrence Chung, B Nixon, Eric Yu, and John Mylopoulos. Non-functional requirements. *Software Engineering*, 2000.

[26] Vittorio Cortellessa, Andrea D'Ambrogio, and Giuseppe Iazeolla. Automatic derivation of software performance models from case documents. *Performance Evaluation*, 45(2):81–105, 2001.

[27] Vittorio Cortellessa and Raffaela Mirandola. Deriving a queueing network based performance model from uml diagrams. In *Proceedings of the 2nd international workshop on Software and performance*, pages 58–70. ACM, 2000.

[28] Edward Curry. Message-oriented middleware. *Middleware for communications*, pages 1–28, 2004.

[29] Miguel de Miguel, Thomas Lambolais, Mehdi Hannouz, Stéphane Betgé-Brezetz, and Sophie Piekarec. Uml extensions for the specification and evaluation of latency constraints in architectural models. In *Proceedings of the 2nd international workshop on Software and performance*, pages 83–88. ACM, 2000.

[30] G Deffenbaugh. Casing the joint. *Unix Review*, 9(12):24–30, 1991.

[31] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 94–103. ACM, 2004.

[32] Dirk Draheim, John Grundy, John Hosking, Christof Lutteroth, and Gerald Weber. Realistic load testing of web applications. In *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, pages 11–pp. IEEE, 2006.

[33] J.E. Durant, SPRC. Software Practices Research Center, and SQE. Software Quality Engineering. *Testing Tools Reference Guide: A Catalog of Software Test & Evaluation Support Tools; Version 12*. Software Quality Engineering, 1991.

[34] Fajar J. Ekaputra, Estefanía Serral, Dietmar Winkler, and Stefan Biffl. An analysis framework for ontology querying tools. In *Proceedings of the 9th International Conference on Semantic Systems*, I-SEMANTICS '13, pages 1–8, New York, NY, USA, 2013. ACM.

[35] Alexander Fay, Stefan Biffl, Dietmar Winkler, Rainer Drath, and Mike Barth. A method to evaluate the openness of automation tools for increased interoperability. In *39th Annual Conference of the Industrial Electronics Society*, pages 6844–6849. IEEE, Nov 2013.

[36] Robert Firth, Vicky Mosley, Richard Pethla, Lauren Roberts, and William Wood. A guide to the classification and assessment of software engineering tools. Technical report, DTIC Document, 1987.

[37] Eric Freeman, Susanne Hupfer, and Ken Arnold. *JavaSpaces principles, patterns, and practice*. Addison-Wesley Professional, 1999.

[38] Vahid Garousi. Fault-driven stress testing of distributed real-time software based on uml models. *Software Testing, Verification and Reliability*, 21(2):101–124, 2011.

[39] Shadi Ghaith, Miao Wang, Philip Perry, and John Murphy. Profile-based, load-independent anomaly detection and analysis in performance regression testing of software systems. In *17th European Conference on Software Maintenance and Reengineering*, pages 379–383. IEEE, 2013.

[40] Stephen Gilmore and Jane Hillston. The pepa workbench: a tool to support a process algebra-based approach to performance modelling. In *Computer Performance Evaluation Modelling Techniques and Tools*, pages 353–368. Springer, 1994.

[41] DR Graham. Computer-aided software testing: The cast report. In *Unicom Seminars Ltd., Middlesex, UK*, 1991.

[42] Vincenzo Grassi and Raffaela Mirandola. Primamob-uml: a methodology for performance analysis of mobile software architectures. In *Proceedings of the 3rd international workshop on Software and performance*, pages 262–274. ACM, 2002.

[43] Object Management Group. Corba services: Common object service specification. Technical report, Object Management Group, 1998.

[44] Gordon P. Gu and Dorina C. Petriu. Xslt transformation from uml models to lqn performance models. In *Proceedings of the 3rd international workshop on Software and performance*, pages 227–234. ACM, 2002.

[45] Reinhard Hametner, Dietmar Winkler, Thomas Östreicher, Stefan Biffl, and Alois Zoitl. The adaptation of test-driven software processes to industrial automation engineering. In *8th IEEE International Conference on Industrial Informatics*, pages 921–927. IEEE, 2010.

[46] Holger Hermanns, Ulrich Herzog, Ulrich Klehmet, Vassilis Mertsiotakis, and Markus Siegle. Compositional performance modelling with the tipptool. *Performance Evaluation*, 39(1):5–35, 2000.

[47] Jane Hillston. *PEPA: Performance enhanced process algebra*. University of Edinburgh, Department of Computer Science, 1993.

[48] Fried Hoeben. Using uml models for performance calculation. In *Proceedings of the 2nd international workshop on Software and performance*, pages 77–82. ACM, 2000.

[49] Gregor Hohpe and Bobby Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[50] Michael N. Huhns and Munindar P. Singh. Service-oriented computing: Key concepts and principles. *Internet Computing, IEEE*, 9(1):75–81, 2005.

[51] Timea Illes, A Herrmann, B Paech, and J Rückert. Criteria for software testing tool evaluation. a task oriented view. In *Proceedings of the 3rd World Congress for Software Quality*, volume 2, pages 213–222, 2005.

[52] ISO Iso. Iec 9126-1: Software engineering-product quality-part 1: Quality model. *Geneva, Switzerland: International Organization for Standardization*, 2001.

[53] Capers Jones. Applied software measurement, 1991.

[54] Pekka Kahkipuro. Uml-based performance modeling framework for component-based distributed systems. In *Performance Engineering, State of the Art and Current Trends*, pages 167–184. Springer-Verlag, 2001.

[55] Manjit Kaur and Raj Kumari. Comparative study of automated testing tools: Testcomplete and quicktest pro. *International Journal of Computer Applications*, 24(1):1–7, 2011.

[56] Peter King and Rob Pooley. Derivation of petri net performance models from uml specifications of communications software. In *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 262–276. Springer, 2000.

[57] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., 1984.

[58] Juhnyoung Lee, Mark Podlaseck, Edith Schonberg, and Robert Hoch. Visualization and analysis of clickstream data of online stores for understanding web merchandising. In *Applications of Data Mining to Electronic Commerce*, pages 59–84. Springer, 2001.

[59] Christoph Lindemann, Axel Thümmler, Alexander Klemm, Marco Lohmann, and Oliver P. Waldhorst. Performance analysis of time-enhanced uml diagrams based on stochastic processes. In *Proceedings of the 3rd international workshop on Software and performance*, pages 25–34. ACM, 2002.

[60] J.D. Meier, Carlos Farre, Prashant Bansode, Scott Barber, and Dennis Rea. *Performance Testing Guidance for Web Applications: Patterns & Practices*. Microsoft Press, Redmond, WA, USA, 2007.

[61] Daniel Menascé. Load testing of web sites. *Internet Computing, IEEE*, 6(4):70–74, 2002.

[62] Daniel A. Menascé and Hassan Gomaa. A method for design and performance modeling of client/server systems. *Software Engineering, IEEE Transactions on*, 26(11):1066–1085, 2000.

[63] Marco Aurélio Stelmar Netto, Suzane Menon, Hugo V. Vieira, Leandro T. Costa, Flavio M. de Oliveira, Rodrigo Saad, and Avelino Zorzo. Evaluating load generation in virtualized environments for software performance testing. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 993–1000. IEEE, 2011.

[64] Benjamin Oberste-Berghaus. Message oriented middleware.

[65] Barbara Paech and Daniel Kerkow. Non-functional requirements engineering-quality is essential. In *10th International Workshop on Requirments Engineering Foundation for Software Quality*, 2004.

[66] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the 4th International Conference on Web Information Systems Engineering*, pages 3–12. IEEE, 2003.

[67] Dorin C. Petriu and Murray Woodside. Software performance models from system scenarios in use case maps. In *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 141–158. Springer, 2002.

[68] Dorina C. Petriu and Hui Shen. Applying the uml performance profile: Graph grammar-based derivation of lqn models from uml specifications. In *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 159–177. Springer, 2002.

[69] Lassi Pohjoisvirta. Choosing a tool for improved software test management. Master's thesis, Tampere University of Technology, 2013.

[70] Rob Pooley. Using uml to derive stochastic process algebra models. 1999.

[71] Rob Pooley and Peter King. The unified modelling language and performance engineering. In *Software, IEE Proceedings-*, volume 146, pages 2–10. IET, 1999.

[72] Robert Poston. The power of simple software testing metrics. *Software Testing Times*, 3(1993), 1990.

[73] Robert Poston. A complete toolkit for the software tester. *Am. Program*, 4(4):34, 1991.

[74] Robert Poston and Michael P. Sexton. Evaluating and selecting testing tools. *Software, IEEE*, 9(3):33–42, 1992.

[75] Tijs Rademakers and Jos Dirksen. *Open-Source ESBs in Action*. Manning Publications, 2008.

[76] Tijs Rademakers and Jos Dirksen. *Open-source ESBs in action*. Manning, 2008.

[77] Frederick F. Reichheld and W. Earl Sasser Jr. Zero defections: quality comes to services. *Harvard business review*, 68(5):105–111, 1989.

[78] Gruia-Catalin Roman. A taxonomy of current issues in requirements engineering. *Computer*, 18(4):14–23, 1985.

[79] Connie U. Smith. Performance engineering of software systems. *Addison-Wesley*, 1:990, 1990.

[80] Connie U. Smith. *Software Performance Engineering. Encyclopedia of Software Engineering*. Wiley, 2002.

[81] Connie U. Smith and Lloyd G. Williams. Performance engineering evaluation of object-oriented systems with spe· ed tm. In *Computer Performance Evaluation Modelling Techniques and Tools*, pages 135–154. Springer, 1997.

[82] CMMI Product Team. Cmmi for development, version 1.2. 2006.

[83] Jiang-Jiang Wang, You-Yin Jing, Chun-Fa Zhang, and Jun-Hong Zhao. Review on multi-criteria decision analysis aid in sustainable energy decision-making. *Renewable and Sustainable Energy Reviews*, 13(9):2263 – 2278, 2009.

[84] Elaine J. Weyuker. Testing component-based software: A cautionary tale. *IEEE software*, 15(5):54–59, 1998.

[85] Lloyd G. Williams and Connie U. Smith. *Performance solutions: a practical guide to creating responsive, scalable software*. Addison-Wesley, Reading, MA, 2001.

[86] D. Winkler, R. Hametner, T. Östreicher, and S. Biffl. A framework for automated testing of automation systems. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–4, Sept 2010.

[87] Dietmar Winkler, Stefan Biffl, and Thomas Östreicher. Test-driven automation–adopting test-first development to improve automation systems engineering processes. In *16th EuroSPI Conference*, 2009.

[88] Dietmar Winkler, Reinhard Hametner, and Stefan Biffl. Automation component aspects for efficient unit testing. In *Proceedings of the conference of Emerging Technologies & Factory Automation*, pages 1–8. IEEE, 2009.

[89] Dietmar Winkler, Thomas Moser, Richard Mordinyi, Wikan Danar Sunindyo, and Stefan Biffl. Engineering object change management process observation in distributed automation systems projects. *Proceedings of 18th European System Software Process Improvement and Innovation (EuroSPI 2011)*, 2011.

[90] Murray Woodside, Greg Franks, and Dorina C. Petriu. The future of software performance engineering. In *Future of Software Engineering*, pages 171–187. IEEE, 2007.

[91] Murray Woodside, Curtis Hrischuk, Bran Selic, and Stefan Bayarov. Automated performance modeling of software generated by a design environment. *Performance Evaluation*, 45(2):107–123, 2001.

APPENDIX A

List of Performance Tools

This Appendix shows the full list of tools and the location where furhter information can be found.

ID	Name	Further Information
1	Allmon	https://code.google.com/p/allmon/
2	Application Testing Suite	http://www.oracle.com/technetwork/oem/app-test/etest-101273.html
3	Appvance	http://http://appvance.com/
4	Benchmark Factory Suite	http://www.questsoftware.de/
5	benerator	http://databene.org/databene-benerator
6	BlazeMeter	http://blazemeter.com/
7	Blitz	https://www.blitz.io/
8	CitraTest	http://www.intelsol.de/
9	CLIF	http://clif.ow2.org/
10	CloudTest	http://www.soasta.com/products/cloudtest-performance/
11	contiperf	http://databene.org/contiperf
12	Curl-loader	http://curl-loader.sourceforge.net/
13	D-ITG	http://traffic.comics.unina.it/software/ITG/
14	Database Open Test Suite	http://ltp.sourceforge.net/
15	DBMonster	http://sourceforge.net/projects/dbmonster/
16	Deluge	http://deluge.sourceforge.net/
17	Dieseltest	http://sourceforge.net/projects/dieseltest/
18	EclipseProfiler	http://sourceforge.net/projects/eclipsecolorer/

ID	Name	Further Information
19	EJP	http://ejp.sourceforge.net/
20	Faban	http://faban.sunsource.net/
21	Funkload	http://funkload.nuxeo.org/
22	FWPTT	http://fwptt.sourceforge.net/
23	Gatling	http://gatling.io/
24	Grinder	http://grinder.sourceforge.net/
25	Grinderstone	https://code.google.com/p/grinderstone/
26	HammerDB (Hammerora)	http://hammerora.sourceforge.net/
27	Hammerhead 2	http://hammerhead.sourceforge.net/
28	HP Load Runner	http://www8.hp.com/us/en/software-solutions/loadrunner-load-testing
29	http_load	http://www.acme.com/software/http_load/
30	httperf	http://www.hpl.hp.com/research/linux/httperf/
31	Hyades (Eclipse TPTP)	http://www.eclipse.org/tptp/
32	IPerf	http://iperf.sourceforge.net/
33	IxoraRMS	http://www.ixorarms.com/
34	j-hawk	http://j-hawk.sourceforge.net/
35	jchav	http://jchav.blogspot.co.at/
36	JCrawler	http://jcrawler.sourceforge.net/
37	JMeter	http://jmeter.apache.org/
38	Jprobe	http://www.infoq.com/news/2008/07/jprobe-8
39	jProf	http://perfinsp.sourceforge.net/jprof.html
40	Jstress	http://sourceforge.net/projects/jstress/
41	JUnit Perf	http://www.clarkware.com/software/JUnitPerf.html
42	JUnit Scenario	http://junitscenario.sourceforge.net/
43	Load Impact	http://loadimpact.com/
44	Load Storm	http://loadstorm.com/
45	loader.io	http://loader.io/
46	LoadSim	http://jobmanager.sourceforge.net/openware_pub/
47	Loadster	http://www.loadsterperformance.com/
48	Load Complete	http://smartbear.com/products/qa-tools/load-testing/
49	Lobo	http://www.oncast.com.br/dev/lobo/index_en.htm
50	LoginVSI	http://www.loginvsi.com/
51	Messadmin	http://messadmin.sourceforge.net/

ID	Name	Further Information
52	MStone	http://mstone.sourceforge.net/
53	Multi-Mechanize	http://multimechanize.com/
54	NeoLoad	http://www.neotys.com/product/overview-neoload.html
55	NGrinder	http://www.nhnopensource.org/ngrinder/
56	NTime	http://www.codeproject.com/dotnet/ntime.asp
57	OpenSTA	http://opensta.org/
58	OpenWebLoad	http://openwebload.sourceforge.net/
59	OptimizeIt	http://www.borland.com/optimizeit/
60	Ostinato	https://code.google.com/p/ostinato/
61	P-unit	http://p-unit.sourceforge.net/
62	P6Spy	http://sourceforge.net/projects/p6spy/
63	PandoraFMS	http://pandora.sourceforge.net/
64	PerfectLoad	Domain no longer reachable
65	PerformaSure	http://discovery.bmc.com/confluence/display/Configipedia/Quest+PerformaSure
66	postal	http://doc.coker.com.au/projects/postal/
67	Pylot	http://www.pylot.org/
68	QACenter	http://www.projectsatwork.com/content/tools/160402.cfm
69	QEngine	http://www.manageengine.com/products/qengine/
70	Rational Performance Tester	http://www.ibm.com/developerworks/downloads/r/rpt/
71	Raw Load Tester	http://www.room4me.com/techtools/RawLoadTester/index.html
72	Seagull	http://gull.sourceforge.net/
73	Siege	http://www.joedog.org/
74	SilkPerformer	http://www.borland.com/products/silkperformer/
75	SimpleProfiler	https://code.google.com/a/eclipselabs.org/p/simpleprofiler/
76	Sipp	http://sipp.sourceforge.net/
77	SLAMD	http://www.slamd.com/
78	Soap-Stone	http://soap-stone.sourceforge.net/
79	SOATest	http://www.parasoft.com/api-testing
80	stress_driver	http://sourceforge.net/projects/stress-driver/
81	Test Load	http://www.histaintl.com/productos/OriginalSoft/partner-microsite/testload.php
82	TestComplete	http://smartbear.com/products/qa-tools/automated-testing-tools/
83	Testing Any-where	http://www.automationanywhere.com/Testing/
84	TestMaker	http://www.pushtotest.com/testmaker-open-source-testing
85	TestStudio	http://www.telerik.com/teststudio

ID	Name	Further Information
86	tivoli monitoring and tivoli composite	http://www-03.ibm.com/software/products/de/tivoli-monitoring-composite-app-mgmt/
87	TPTEST	http://tptest.sourceforge.net/about.php
88	Tsung	http://tsung.erlang-projects.org/
89	Valgrind	http://valgrind.org/
90	Visual Studio	http://www.visualstudio.com/
91	WAPT	http://www.loadtestingtool.com/
92	Web Polygraph	http://www.web-polygraph.org/
93	WebLOAD	http://radview.com/ or http://www.webload.org/
94	App Perfect	http://www.appperfect.com/products/load-test.html
95	Apache Flood	http://httpd.apache.org/test/flood/

Table A.1: Available Tools

The sources that this list is based on, are the search at multiple search engines and summarizing pages, such as: http://en.wikipedia.org/wiki/Load_testing
http://www.opensourcetesting.org/performance.php
http://www.softwaretestinghelp.com/performance-testing-tools-load-testing-tools/
https://www.testtoolreview.de/de/testing-tools/tool-liste?filterreset=true&categories%5BLoad-/Performance-Test%5D=10

List of Figures

List of Tables