Testing of Web-based Systems

Authors

Stian Frydenlund Lereng
Lars Fugelseth

Supervisor

Tor Stålhane

“Do not anticipate trouble, or worry about what may never happen. Keep in the sunlight.”

- Benjamin Franklin -
Abstract

Web-based systems represent a young, but rapidly growing technology. As the number of web applications continues to grow, these systems enter a critical role in a multitude of companies. The way web systems impact business aspects, combined with an ever-growing internet user mass, emphasize the importance of developing high-quality products. Thus, proper testing plays a distinctive part in ensuring reliable, robust and high-performing operation of web applications.

In this report we hope to uncover some of the differences that distinguish web-based systems from traditional software applications, and how they, in turn, influence the testing process. Then we take a look at relevant quality attribute definitions and how reliability, robustness and performance can be categorized to shape the structure of our report. With the quality attribute interpretations in place, testing strategies and considerations are presented for reliability, robustness and performance. Finally, we perform an informal evaluation based on hands-on experience with two comprehensive software tools for automated functional testing of web applications. We assess what core functionality they offer and how they utilize HTML resources for testing purposes.
Preface

This report is the product of a specialization project in TDT4735 Software Engineering at the Department of Computer and Information Science. The “Testing of Web-based Systems” project has been performed by two master students – Lars Fugelseth and Stian Frydenlund Lereng – during our 5th and final year at the Norwegian University of Science and Technology.

The assignment was proposed by professor Tor Stålhane, who gave us freedom to define the scope of the task ourselves. Thus, we decided to combine available literature with our own experiences, interpretations and thoughts to determine web system characteristics and to explore how vital application properties such as reliability, robustness and performance, can be tested and achieved. We also found it feasible to look at a few selected automated testing tools, in order to attain a basic understanding of how automated functional testing can be performed in a web context.

We would like to thank our supervisor Tor Stålhane for taking the time to listen to our ideas, for providing us with relevant information to get started, and for giving valuable feedback in the later stages.

We would like to compliment Seague’s Nordic distributor – the Danish MCG Software – who has shown genuine interest in assisting our process of getting to know SilkTest and has encouraged us to contact them whenever problems or questions arose, customer relation manager Beate Richter in particular. Janis Stenset at Mercury’s Centralised Sales Division also deserves credit for providing us with product information.

Trondheim, 25th of November

_____________________________
Stian Frydenlund Lereng

_____________________________
Lars Fugelseth
# Table of Contents

CHAPTER 1: Introduction................................................................. 1  
1.1 Motivation ............................................................................. 1  
1.2 Project Context .............................................................. 2  
1.3 Problem Definition ......................................................... 2  
1.4 Project Scope ................................................................. 3  
1.5 Project Goals ................................................................. 3  
1.6. Term Clarification............................................................ 4  
1.7 Report Outline .................................................................... 4  

CHAPTER 2: Distinctions and Elements of Web Applications .......... 6  
2.1 Development, Technology and Scope ........................................ 6  
2.1.1 Development .................................................................. 6  
2.1.2 Technology .................................................................. 7  
2.1.3 Scope ........................................................................... 8  
2.2 Challenges of Testing Web Applications ................................. 8  
2.2.1 Development and Technology Complicates Testing .......... 9  
2.2.2 The Importance of Web Testing from a Business Viewpoint ... 9  
2.3 Web Services ....................................................................... 10  
2.3.1 Functional Testing and Load Testing ............................... 12  
2.3.2 Scalability Issues and Considerations ......................... 12  
2.4 Traditional Software Testing Techniques ............................... 12  
2.4.1 Purpose of Testing .......................................................... 13  
2.4.2 Unit, Integration and System Testing ............................... 13  
2.4.3 Black Box and White Box Testing ................................. 14  
2.4.4 How to Counteract Errors and Bugs .............................. 15  
2.5 Outsourcing Web Testing .................................................. 16  
2.5.1 When to Consider Outsourcing Testing Activities .......... 16  
2.5.2 Business Considerations to Be Made ............................. 16  
2.5.3 The Pros and Cons of Outsourcing Web Testing .......... 17  

CHAPTER 3: Categorization and Interpretations of Quality Attributes... 18  
3.1 Mapping of Quality Attributes to ISO 9126 ............................. 18  
3.2 Grouping of Quality Attributes ........................................... 19  
3.2.1 Reliability versus Robustness ....................................... 20  

CHAPTER 4: Reliability and Robustness Testing ............................. 22  
4.1 Reliability Testing ............................................................. 22  
4.1.2 Reliability Definitions ................................................... 22  
4.1.2. Reliability Goals and Challenges .................................. 23  
4.1.3 Reliability Testing of Web Applications ........................... 23  
4.1.3.1 Component Stress Testing ........................................... 24  
4.1.3.2 Integration Stress Testing .......................................... 24  
4.1.3.3 Real-world Testing ................................................... 25  
4.1.3.4 Random Destruction Testing .................................... 25  
4.1.3.5 Quality Assurance Testing ...................................... 25  
4.1.4 An Alternative Strategy for Reliability Testing ............... 26  
4.1.4.1 Strategy Foundation and Definitions .......................... 26  
4.1.4.2 Usage Models and Key Components ........................... 27  
4.1.4.3 Operational Profile Construction and Log Analysis ...... 27  
4.1.4.4 Strategy Implementation and Reliability Assessment .... 28  
4.1.4.5 Results ................................................................. 29
# Table of Contents

4.2 Robustness testing ................................................................. 30  
4.2.1 Robustness Definitions ..................................................... 30  
4.2.2 Why Test Software Robustness? ........................................ 30  
4.2.3 Robustness Testing of Web Applications ......................... 31  
  4.2.3.1 Error Tolerance ........................................................... 32  
  4.2.3.2 Stress Tolerance ......................................................... 33  
  4.2.3.3 Platform Tolerance ...................................................... 34  

CHAPTER 5: Performance testing .................................................. 37  
5.1 Goals and Basics of Performance Testing ......................... 37  
  5.1.1 Performance Measures and Output .................................. 37  
  5.1.2 Primary Performance Factors ........................................ 38  
5.2 Performance Testing Techniques .......................................... 38  
  5.2.1 Load Testing ................................................................. 38  
  5.2.2 Stress Testing .............................................................. 39  
  5.2.3 Capacity Testing .......................................................... 39  
  5.2.4 Definition of Key Terms and Measures ......................... 39  
5.3 Workload ............................................................................. 41  
  5.3.1 Workload Modelling ....................................................... 41  
    5.3.1.1 Key Scenarios .......................................................... 41  
    5.3.1.2 Maximum User Amount Expected ............................ 41  
    5.3.1.3 Possible User Actions .............................................. 42  
    5.3.1.4 Existing User Profiles ............................................. 42  
    5.3.1.5 User Profile Operations .......................................... 42  
    5.3.1.6 Average Think Time .............................................. 42  
    5.3.1.7 Expected User Profile Composition ....................... 42  
    5.3.1.8 Test Duration ........................................................ 42  
5.4 Performing Load Testing ...................................................... 42  
  5.4.1 Step 1 – Identify Key Scenarios .................................... 43  
  5.4.2 Step 2 – Identify Workload ............................................ 43  
  5.4.3 Step 3 – Identify Metrics ............................................... 44  
  5.4.4 Step 4 – Create Test Cases .......................................... 44  
  5.4.5 Step 5 – Simulate Load ............................................... 44  
  5.4.6 Step 6 – Analyze Results ............................................. 44  
  5.4.7 Benefits of Load Testing ............................................... 45  
5.5 Performing Stress Testing .................................................... 45  
  5.5.1 Step 1 – Identify Key Scenarios .................................... 46  
  5.5.2 Step 2 – Identify Workload ............................................ 46  
  5.5.3 Step 3 – Identify Metrics ............................................... 46  
  5.5.4 Step 4 – Create Test Cases .......................................... 46  
  5.5.5 Step 5 – Simulate Load ............................................... 46  
  5.5.6 Step 6 – Analyze Results ............................................. 47  
5.6 An Alternative Outlook on Performance Testing .................. 47  
  5.6.1 Planning Phase ............................................................ 47  
  5.6.2 Testing Phase .............................................................. 47  
  5.6.3 Analysis Phase ............................................................ 47  
5.7 Guidelines for Successful Performance Testing .................... 48  
  5.7.1 Environmental Considerations ..................................... 48  
  5.7.2 Simultaneous and Concurrent User Considerations .......... 48  
  5.7.3 Miscellaneous Considerations ..................................... 49  
5.8 Analysis of Performance Data .............................................. 49
CHAPTER 6: Evaluation of Automated Testing Tools for Web Applications ......................................................... 52
6.1 Evaluation Criteria ................................................................................................................................. 52
6.2 Target Web Application .................................................................................................................... 52
6.3 Installation and Learnability .............................................................................................................. 53
   6.3.1 Mercury QuickTest Professional 8.0 ......................................................................................... 54
   6.3.2 Segue SilkTest 7.1 ..................................................................................................................... 54
6.4 Testing of a Web-based System for Performing Surveys with Mercury QuickTest ................................................................. 55
   6.4.1 Element Identification ................................................................................................................. 55
   6.4.2 Verification Through Checkpoints ............................................................................................ 57
6.5 Testing of a Web-based System for Performing Surveys with Segue SilkTest ................................................................. 58
   6.5.1 Creating a Basic Test .................................................................................................................. 59
   6.5.2 Constructing a Data-driven Test ............................................................................................... 59
   6.5.3 Element Identification ................................................................................................................. 60
6.6 Type of Errors Detected .................................................................................................................... 60
6.7 Coverage .......................................................................................................................................... 61
6.8 Test Set Generation ............................................................................................................................ 61
6.9 Reporting and Presentation of Test Results ....................................................................................... 62
6.10 Time Consumption ........................................................................................................................... 63
6.11 Multi-platform Support .................................................................................................................... 64
6.12 Summary .......................................................................................................................................... 65
   6.12.1 Tool Similarities ....................................................................................................................... 65
   6.12.2 Tool Differences ....................................................................................................................... 65
   6.12.3 Final Comments ....................................................................................................................... 68
CHAPTER 7: Conclusion .............................................................................................................................. 67
7.1 Summary .......................................................................................................................................... 67
7.2 Further work ...................................................................................................................................... 68
Appendix A ............................................................................................................................................ 69
Appendix B ............................................................................................................................................ 71
Appendix C ............................................................................................................................................ 77
List of Figures

Figure 2.1 Web system architecture 7
Figure 2.2 XML Web Service example 11
Figure 2.3 Test composition example 14
Figure 2.4 An illustration of black box and white box testing 15
Figure 3.1 Software quality attributes in ISO 9126 18
Figure 3.2 An alternative quality attribute structure 19
Figure 3.3 Reliability defined by ISO 9126 and robustness defined by [SZ03] 20
Figure 4.1 The four steps of Microsoft’s reliability testing strategy 24
Figure 4.2 Hierarchical implementation 29
Figure 4.3 The relationship between reliability and robustness 31
Figure 5.1 Process blueprint for load and stress testing of web applications 43
Figure 5.2 Throughput versus user load 49
Figure 5.3 Response time versus user load 50
Figure 5.4 Processor utilization versus user load 50
Figure 6.1 Survey construction 53
Figure 6.2 Survey overview 55
Figure 6.3 Dialog box for element identification 56
Figure 6.4 Dialog box for specifying element values in SilkTest 58
Figure 6.5 SilkTest functionality for replacing test values with Excel spreadsheet data 59
Figure 6.6 Presentation of test results in QuickTest 62
Figure 6.7 Presentation of test results in SilkTest 63
Figure 6.8 SilkTest interface displaying resulting code from recording 67
Figure 6.9 QuickTest’s graphical test representation, along with the built-in spreadsheet and browser 68
List of Tables

Table 2.1: Summary of web system features and challenges 8
Table 3.1: Efficiency definition, ISO 9126 19
Table 4.1: Reports based on log analysis 28
Table 5.1: Goals of load, stress and capacity testing 39
Table 5.2: Metric classification 44
Table 6.1: Summary of tool similarities 65
Table 6.2: Summary of important tool characteristics 66
CHAPTER 1: Introduction

This introduction section will briefly state the main motivation and objectives related to the project we have been assigned: “Testing of Web-based Systems”. Further we explain the project context, try to define the most significant problems as far as web-based applications are concerned, and highlight the direction that we have chosen within this broad theme. Finally we put our goals for the project in concrete terms and give a presentation of the report outline and what the different chapters will contain.

1.1 Motivation

In recent years web applications have come to play an increasingly important part in software engineering in general, and also play a vital part in our everyday lives. As these systems offer more and more functionality to the user and enter new markets and new business segments, testing of such systems becomes all the more relevant, important and challenging. Being a relatively young technology there is a risk that existing tools and methods are either imperfect or simply have not been employed by certain software vendors. The lack of experience in testing web applications might be one reason why some companies find it tempting to use ad hoc testing techniques and even make testing the big loser when time is running out.

Testing becomes an even more important factor when we consider the numerous properties of web systems that we do not necessarily find in traditional software applications. We will address this, and highlight the differences between traditional systems and web-based systems, in the following chapter. The overwhelming number of users on the World Wide Web today indirectly puts an emphasis on the importance of well-tested web applications, to avoid the user encountering problems or experiencing any kind of dissatisfaction. The competition for potential customers and clients is increasing and testing might well prove to be one of the key factors for attaining success.

Another thing that puzzled us was the fact that although finding literature on web-based systems in general was not difficult, they rarely or only superficially dealt with testing of such systems. And whereas there exist numerous books on software testing they tend to neglect web applications. With all these challenges facing the testing of web-based systems we saw it as an interesting and highly relevant task, and also an opportunity to dive
into a subject that is very much of current interest to the software community.

1.2 Project Context

This project was assigned to us as a specialization project in software engineering by the department of software engineering at the Norwegian University of Science and Technology (NTNU). Neither of us has any prior, specific knowledge of, nor experience with, organized, formal testing of software. The only theoretical foundation and hands-on experience we possess with regards to software testing comes solely from previous software engineering courses and projects. The “Testing of Web-based Systems” project is related to the ongoing WebSys project at the same institute, but there is no direct connection to this or any other long-term project.

The time frame for this project does not allow us to go in depth in every single area of the subject. This implies that we had to restrict our focus to chosen parts of the problem domain, and the resulting project scope is elaborated in section 1.4.

1.3 Problem Definition

The problems and challenges that come in the wake of testing web systems are many and diverse. First of all there seems to be inadequate technical expertise in the field and also a lack of mature testing tools for these applications. The main cause for this is most likely web systems, and the testing of these, being a relatively “juvenile” and rapidly changing technology. The temptation for software vendors to develop web applications with ad hoc software engineering processes might also lead to a somewhat informal version management, thus complicating testing matters by not having a properly working version to revert to. This makes analyzing defects even harder since one has to deal with an environment that is becoming increasingly complex.

There are also testing concerns from a more technical point of view. Web systems are built on partly separate development technologies related to programming language, user interface and user interaction. They depend on COTS products – Commercial Off-The-Shelf – that can not be fully controlled, such as Flash, Java Applet, Acrobat Reader and ActiveX to name a few. Multiple users sharing the same resources, like a web server, increases the possibility of the user experiencing some kind of trouble, especially when the number of users wanting to use a particular service exceeds the threshold for what the system can handle.

Another complicating factor is the pressure of making time-to-market as short as possible. This has led the web industry into shorter iterations, implying that test passes ought to or must be performed in step with the development cycles. This in turn means that testing activities will take place
more frequently than what is the case for traditional software engineering. The tests to be performed will have to focus on detecting errors that may cause unacceptable levels of robustness. The latter rises the challenge of finding methods tailored to testing of web systems and assess their suitability for the robustness of such applications.

All of the issues mentioned above, combined with the multitude of different browser versions among different vendors, make testing and quality assurance in the context of web-based systems a challenging prospect on more than one level.

1.4 Project Scope

In this project we want to investigate how testing of web-based systems is performed. Further, we want to focus on two main areas within web software. The first area is concerned with robustness and reliability of such systems, while the second puts the emphasis on performance. Our goal is to find and evaluate two selected software-based tools that perform automated functional testing of web systems, and possibly compare them.

To narrow down the task at hand even further, we have decided to concentrate on systems that interact with the user, mainly applications that make use of a central database for information storage, where user input plays an important part in the total operation of the system. We believe that it is within these types of applications that user actions are destined to have consequences and thus challenge the robustness of the system in use.

1.5 Project Goals

We have decided to devote a separate section of the introduction to describe our main goals for the project, because we believe that these goals will contribute to the understanding of why this project ended up the way it did and why we made the decisions we made during the process. The most obvious goal is perhaps that we want to acquire knowledge about how testing of software in general is performed, but we also have more specific expectations to the project as a whole.

Initially we hope to uncover the main challenges related to testing of web-based systems, and compare testing of such applications to the testing of “regular” software. This will give us a broad understanding of the topic and build a platform for exploring chosen aspects in detail. We also aim at finding out how and when web testing activities can be outsourced. The focus as far as this point is concerned, rests on how a software business chooses to deal with the testing discipline; whether to do all the testing themselves or do parts of the testing within the organization and let external companies that have a specialization within the domain take care of most of it, if not all.
Finally we want to familiarize ourselves with a couple of automated testing tools and thus get to know their abilities, possibilities and limitations. Our initial impression is that the market has a lot of testing tools to offer, but that they tend to cover the same software properties or quality attributes, while neglecting others. With this in mind we hope to find out if the automated functional testing offered by our selected tools contributes to addressing robustness issues in particular.

1.6. Term Clarification

Throughout this report we have used the terms web-based system, web system and web application interchangeably. The motivation for this has been to add language variety and thus avoid repeating one specific term over and over again. At times we have merely used system or application in the meaning of web-based system. Systems that are not web-based are referred to as a traditional or regular applications or systems.

1.7 Report Outline

We have chosen to let the opening chapters deal with general aspects of web-based systems and testing of these, while progressively getting more specific as the report goes on. Hence the first three chapters are rather broad in their content and can be seen as an introduction or build-up for the subsequent three chapters. The intention is to build a reference or context for chapters 4 through 6, which deal with the core issues of this assignment.

Chapter 1 gives an introduction to the project, defining its context, motivation and scope and explaining our goals for the assignment.

Chapter 2 takes a look at the characteristics of web applications and how these separate web-based systems from traditional software, both with respect to development and testing.

Chapter 3 seeks to explain the relationship between our chosen quality attributes, how they map to existing quality models and how this has influenced the structure of chapters 4 and 5.

Chapter 4 deals with reliability and robustness testing of web-based systems. We will look at definitions and challenges of both reliability and robustness and present ways of testing each of them.

Chapter 5 focuses on performance testing and strives to explain how load, stress and capacity testing contribute to the assessment of a web application’s performance.

Chapter 6 presents two automated testing tools for web systems – Segue SilkTest version 7.1 and Mercury QuickTest Professional version 8.0. We
have evaluated these with respect to a set of criteria and will elaborate on our experiences with each of them.

Chapter 7 rounds off the report by drawing conclusions, describing lessons learned, and looks at the possibility of further work.
CHAPTER 2: Distinctions and Elements of Web Applications

This chapter aims to reveal some of the aspects that separate web applications from regular software. Some of the differences related to development and technology were touched upon in the introductory chapter, but this chapter takes the subject a bit further. We will also look at what challenges these differences will lead to when it comes to testing of both web applications and web services. There is also a small section devoted to traditional and well-known software testing techniques. Finally the chapter is rounded off with a paragraph on outsourcing testing activities of web-based systems.

2.1 Development, Technology and Scope

As mentioned in the opening parts of this chapter, a few elementary differences were brought up in the introduction. In this section we have tried to group some of the main differences into three categories to improve the structure and thus make it easier to get an overview.

2.1.1 Development

One key feature of the development process for web-based systems is the presence of so-called fuzzy requirements that are heavily driven by factors such as layout and advertising. There is a greater focus on graphical user interface, from now on referred to as GUI, and as a result GUI plays a far more commanding role in the overall development process than what one would find in standard software applications. With GUI being one of the main drivers during development we could say that the creation of web systems pays more attention to the "how to do it" than what we are used to from traditional software engineering. The distinctive nature of web applications demands a vast variety of expertise, thus involving webmasters and people with valuable know-how in graphics design and marketing, just to name a few.

Web applications and their design, layout and content also tend to change more frequently and rapidly. Even though the changes occur at a frequent rate they are usually not dramatic, nor do they make too much of an impact most of the time. However, there is a risk of running into trouble, leading to serious consequences if the changes made are what Coast Web Quality
Distinctions and Elements of Web Applications

Management refers to as unmanaged change in “Web Quality Testing and the Rational Unified Process” [CO02]. The problem or challenge is that the minor updates are performed in real-time and the person implementing the changes is typically an employee of the organization who has had little or nothing to do with the development of the system in use.

As a result of rapid changes and shorter time frames, developing web systems is in most cases an iterative process, with short and intensive iterations taking place. However, what makes this aspect of web application development even more special is the fact that the system to be transferred to the production site is actually a prototype. This last point does not match well with most traditional software engineering paradigms we have learnt over the years.

2.1.2 Technology

The first thing that probably strikes most of us when given the question of what distinguishes a web application from other software, is that these applications are accessed through a browser. And even though the use of a browser for any type of web system might suggest a standardized and less complex way of dealing with software, the myriad of browser versions among the different vendors complicates testing matters as well as the general quality assurance.

Another distinct feature of web pages is the hyperlink, offering the possibility to bring the user to a different site or to a particular web page. Furthermore web systems do not require any installation in the way we are used to think of that term. With the continuously updated prototype "shipped" to the production site and with all access or user interaction with the application taking place via web browsers, the installation process is practically non-existent.

![Figure 2.1 Web system architecture](Figure 2.1)
Not only do web applications involve several third party products, whose challenges will be mentioned in section 2.2 of this chapter, but they also have several physical points where potential slow-downs might occur. A typical three-tiered web system is shown in figure 2.1 above. Hence the performance of most web-based systems nowadays relies heavily on web servers, application servers and database servers, according to the article “The Living Creature – Testing Web Applications” by QA Labs Inc. [QA00].

2.1.3 Scope

We mentioned the prospect of minimizing time-to-market briefly in the introduction. This has had a significant impact on the time frame of today’s web projects, where developers try hard to meet the demands and wishes of the marketing department, knowing that time is money. This has effectively led to an average time frame of four months, compared to up to three years for an ordinary software development project [QA00]. The gap between these types of projects is a contributing factor in explaining the differences taking place during the entire development stage.

The difference in the amount of potential users is also important. Not only is a system on the web likely to have far more users than a standard application used internally within a business, but that user mass is likely to vary a lot and also be hard to assess or predict with acceptable accuracy. On top of all that, the level of general computer understanding and software knowledge among potential users is destined to fill the entire spectre – from the inexperienced novice to the computer expert. As opposed to traditional software applications used within organizations, there is no formal training to make sure that the users have reached a certain level.

<table>
<thead>
<tr>
<th>Development</th>
<th>Technology</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzy requirements</td>
<td>Browser</td>
<td>Time-to-market</td>
</tr>
<tr>
<td>GUI focus</td>
<td>Links</td>
<td>Development time</td>
</tr>
<tr>
<td>Expertise from different fields involved</td>
<td>No installation</td>
<td>Unstable and hard-to-predict user amount</td>
</tr>
<tr>
<td>Frequent and rapid changes</td>
<td>Web server, application server, database server</td>
<td>Variety in user skills</td>
</tr>
<tr>
<td>Iterative development</td>
<td>Third-party products</td>
<td></td>
</tr>
<tr>
<td>Prototype</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Summary of web system features and challenges

2.2 Challenges of Testing Web Applications

Quite a few testing challenges were superficially described in the problem definition of the introductory chapter, but this section will elaborate on those issues and mention a few new ones as well. Many
of the web testing challenges are strongly related to the web system features and differences brought to the surface in the previous section.

2.2.1 Development and Technology Complicates Testing

Some testing difficulties arise as a result of the way web applications are developed. According to [QA00] there is a tendency that some functionality, often thought of as supplementary, is added to the system after the web site is up and running and thus available to the public. The process of adding functionality as new ideas pop up, logically complicates testing matters. Performing software engineering in an informal manner, based on loose expectations and wishes of necessary functionality will not yield an environment where requirements are stable. The latter is essential in order to perform thorough testing and is usually the case in traditional software projects where testing is done based on a requirement specification. Thus, it is pivotal from a testing point of view that requirements specification is done when developing web-based systems.

As mentioned in chapter 2.1.1 an evolutionary development model is the most likely candidate framework for web applications development. With short and intense iterations it is important that a testing session is performed before the end of each iteration. Building increments without putting sufficient testing effort into them is destined to affect the overall quality of the end functionality. Each test undertaken should operate on a browser level in order to make testing as realistic as possible, seeing how future users will access the system via web browsers. When testing performance or making performance measurements, the application should not be executed locally, since this would not yield realistic or relevant results.

The involvement of third-party products such as software related to SSL, IIS, Apache and the likes, creates extra complexity and importance as far as testing is concerned. In particular [QA00] mentions the fact that different security mechanisms used to verify users and regulate access to information make web applications vulnerable. This represents an opportunity to those who might want to explore and take advantage of errors and weaknesses in third-party products and thereby get access to servers, databases or scripts. Thus, testing of different security mechanisms and how the system interacts with them can not be neglected.

2.2.2 The Importance of Web Testing from a Business Viewpoint

Usability, a sub-attribute of usefulness, represents a challenging field both on and off the web. With more and more focus on usability, testing of this quality attribute becomes all the more important. It is, however, logical to claim that usability issues weigh even more when it comes to web systems, because the potential user mass is huge and the experience in using the internet and web applications varies tremendously. Most online businesses will seek to embrace a broad audience and this implies that feedback from potential users is invaluable, and the sooner they can get that feedback the
sooner they can adjust the course, possibly before releasing the final version. There are two alternatives to get the needed feedback according to [QA00]:

1) Capture and quantify the meaning of the terms learnability, understandability and operability in a form that is testable
2) Create a group of representative target users and observe them as they use the web application

Even though usability is an interesting and important topic from a web perspective, this is not a quality attribute we have chosen to emphasize, thus we will not elaborate any further on this.

It sounds reasonable to assume that testing might play an even more critical part in a web context knowing that the consequences of poor quality might be even bigger for companies running e-business or e-commerce applications, who have the company web site and its underlying functionality as the main show window for potential customers. In such systems the quality attributes that we have chosen – robustness, reliability and performance – become important properties, more so than in most traditional software. Making sure the web application is robust is of utmost importance, knowing that the user can be anything from a first-time internet visitor to a computer or software expert, none of whom will have had any specialized training in using that particular application. Whereas robustness is mostly geared towards the inexperienced web users, reliability and performance affect everyone. An employee using a legacy system in his organization will most likely not have an alternative system to use if he or she is unhappy with the performance or reliability of the system in use, and therefore has to tolerate it. A customer surfing the web with a certain goal in mind knows he does not have to tolerate lack in performance or unacceptable reliability, because there is likely to be a multitude of other companies offering similar services just a mouse-click away. Thus, the web user will have a lower threshold as far as patience and tolerance is concerned, which highlights the importance of making sure web-based systems are reliable and perform satisfactory.

2.3 Web Services

It is hard to talk about web-based systems these days without mentioning web services. It is expected that more functionality will be transferred to web services while browser-dependent parts of the system will mainly serve interface purposes. Thus, web services are seemingly becoming building blocks for future web applications and are integrated in many of today’s web-based bank applications and online booking systems. In an article titled “Web Service Testing” by Red Gate Software Ltd. [Dav02], Neil Davidson refers to flight booking systems where a web server will try to access a web service and have that service process the request instead of processing database information directly. However, being part of the internet poses a challenge to any web service since it is accessible for
1. Development of XML Web Service
   Developer creates an XML Web Service for sending SMS messages to cellular phones.

2. Publishing of XML Web Service
   Developer makes the service known by publishing the WSDL document using UDDI.

3. XML Web Service Discovery
   Developer searches through UDDI and selects an XML Web Service for sending SMS messages to cellular phones.

4. XML Web Service Interaction
   The web application and the XML Web Service interact using the SOAP messages defined in the published WSDL document.

Figure 2.2
XML Web Service example
everyone, and this forces us to consider scalability and security aspects. Figure 2.2 depicts a usage scenario for an XML web service.

### 2.3.1 Functional Testing and Load Testing

As far as testing of web services is concerned [Dav02] stresses the importance of automated testing, particularly because web services do not have any user interface. Functional testing and load or stress testing need most effort. Besides the goal of checking that the web service’s functionality meets the requirements set forth, functional testing should also verify bounds and detect possible errors. Load or stress testing is important in order to find out how well that particular web service copes with an increasing amount of clients trying to access it. Such testing will determine the quality attribute of scalability, which is an important property in web-based systems in general. Stress testing must according to [Dav02] be an automated task, because of the nature of web services.

### 2.3.2 Scalability Issues and Considerations

As mentioned in the previous section, scalability plays a huge part in any web service’s operation. Monitoring how the web service responds when the number of clients increases is pivotal, and requires that all other factors that might also affect the overall scalability is kept constant. [Dav02] lists three key values that need to be identified as the amount of users changes:

- Connection time from client to web service
- Time elapsed before client receives data from the web service
- Time elapsed before client receives last byte of information

How these three values change as the rate of clients accessing the web service varies needs to be monitored. In an ideal situation all three factors would stay close to constant, and that is what any web service will strive for. However, the most likely scenario facing us would be a web service scaling well until a certain client threshold is reached, and when this limit is exceeded, the general performance will drop. The challenge is to discover possible bottlenecks and find a solution to the problems. Perfection will be hard to attain, and thus one has to keep in mind which values are acceptable and which values are not. Response times for a web service can not be compared to those of a web site.

### 2.4 Traditional Software Testing Techniques

This section will give a rough overview of the software testing techniques that are common in traditional software development. The motivation behind this is to get acquainted with the terms in general and to be able to refer to the methods described here later in the project report. Besides, the main purpose of testing is a conception shared by all types of software, including web-based systems.
2.4.1 Purpose of Testing

The goal of all testing is to demonstrate that the program under scrutiny contains bugs. Testing must not be confused with debugging, which is the process of detecting and reducing the number of existing errors. Testing can never prove that a code is error-free, but rather verify that errors exist. No matter what testing technique is being used testing should be an activity considered thoroughly and a test design ought to be made before any programming takes place. The reason for this is that the creation of a test design will often lead to detection or prevention of bugs, which saves both time and money. A test design contains expectations as to the result we will get from the tests, a description on how the tests will be carried out and the actual outcome of the tests. All of the points just mentioned can possibly contain errors, therefore we have to consider the fact that the error might come from the test itself, while the tested code might be correct. We have to know what the results of the test will show before it has actually been performed. Whoever is responsible for doing the testing has to be able to define what the outcome should be, if not, this will lead to bugs in either the program or the test or in both the program and the test. The good thing about software testing is that it can be carried out without any prior knowledge about the program design and can thus be performed by "outsiders".

Another possibility is to make the testing process automatic where suitable. For instance, we could automate measurements of a function’s or method’s input producing the correct output. The tests carried out will provide developers and testers with symptoms that there exists a problem, by showing a mismatch in expected results and actual results. In order to find out exactly what causes the problem, more detailed testing might prove necessary. In general the need for testing increases with the size of the application. One way to reduce the code complexity is to split up the application into several modules, but this increases the complexity and risk for bugs in the interfaces between the modules.

2.4.2 Unit, Integration and System Testing

Boris Beizer’s “Software Testing Techniques” [Bei90] groups testing into three classes where different testing techniques are employed and being carried out in different stages of the software development process:

- Unit testing, also referred to as component testing
- Integration testing
- System testing

Unit testing consists of testing a small piece of software code made by a programmer, whereas component testing involves testing of components consisting of one or more units. The test is meant to show whether or not the code satisfies the functional requirements and whether the structure suits the design structure.
When components are being connected to create larger components they have to pass through integration testing, whose main purpose is to detect any inconsistency between the connected components. Once the integration test is completed, a component test will have to be performed on the new component, consisting of several smaller ones. Some properties, however, can be said to belong to the system as a whole, like certain quality attributes. The system, being a huge component, will hence be tested in its entirety to verify that these requirements are met in a satisfactory way, and this is what is referred to as system testing. Figure 2.3 illustrates the sequence of testing, starting with unit testing, then moving on to integration testing before component testing can be performed. Finally, a system test is executed.

2.4.3 Black Box and White Box Testing

When a tester wants to test an application or a particular part of it to detect bugs, he can look at the system from a user’s perspective and expose it to different types of input, and then check whether or not the resulting output is in accordance with the specification. This is known as functional testing or black box testing. With this type of testing, any type of error can be detected, but it would take an infinite amount of time to do so [Bei90]. This statement is, however, contradicted by Brian Bryson from IBM Rational, in his article “Bridging the Gap Between Black Box and White Box Testing” [Bry03]. He claims that black box testing is not capable of detecting all
types of errors and refers to memory leaks as a typical example of an error that will escape functional testing.

Structural testing or white box testing, on the other hand, implies studying the program code and testing the different parts of it. This type of testing is not capable of finding all kinds of errors, but to its advantage it is easier to determine when you have tested enough [Bei90]. Then again, [Bry03] states that white box testing is capable of testing all lines of code when given an infinite amount of time, but points out that such testing might not uncover errors with respect to component integration. Thus, [Bei90] and [Bry03] seem to contradict each other. In spite of this, our interpretation suggests that both of them find a combination of white box and black box testing appropriate, in order to maximize error detection. A comparison between the two testing techniques is shown in figure 2.4.

2.4.4 How to Counteract Errors and Bugs

There are several lines of action in uncovering and preventing bugs from sneaking into a software program. [Bei90] mentions inspection methods, program design and design methods, among others, as ways of counteracting possible bugs entering software programs. However, there will always be errors in software applications because they tend to grow more complex as time passes by, and up until this date human beings have no method of insuring flawless communication that will effectively eliminate any misunderstanding. We also need to keep in mind that there is no way of proving that a program is free of faults. However, testing is an activity that enables us to say something about the probability of a hidden error creating problems.
2.5 Outsourcing Web Testing

With testing being a demanding and resource-consuming activity, the option of outsourcing web testing has become a reality. Companies whose web systems are essential for their business are aware of the significance of proper testing in order to attain quality products, and therefore believe that testing should be done by competent personnel.

2.5.1 When to Consider Outsourcing Testing Activities

“Outsourcing Web Site Testing” by Derek Sisson [Sis02] describes potential reasons for a company to outsource web testing activities. One of them refers to the scenario that there simply are not sufficient resources within the organization to perform the testing themselves. Outsourcing becomes an even more likely option if all or most of the testing exercises can be relegated to the same contractor.

If the development project is estimated to have a relatively short lifetime, it makes sense for businesses to outsource testing activities instead of investing money in extensive testing processes that they strictly speaking do not need in the long run. In addition, outsourcing will provide a neutral view on the overall quality of the product from an objective third party, which might give useful input to the development team. Some companies even choose to outsource programming and general development tasks. In such cases, it is meaningful to also outsource testing of the web application, since the contractor already possesses knowledge about the technical aspects of the system and thus has a better basis for executing tests.

2.5.2 Business Considerations to Be Made

One of the main things that [Sis02] stresses is the fact that the development organization, being the owner of the application, needs to know the exact scope of the tasks considered to be outsourced. Understanding the scope of the task at hand makes it considerably easier to deal with the expectations and gives everyone involved a more realistic picture on the outcome. In addition, the company needs to understand what tasks are involved in testing the system and how this influences the total quality of the end product. However, outsourcing testing activities is not the same as outsourcing quality assurance as a whole. Testing is simply a tool or a method used within the boundaries of quality assurance. One could say that proper and thorough testing is a necessity in achieving quality, but it is not sufficient to guarantee it.

Misunderstandings between the developing organization and the contractor are destined to prove costly, hence the company must make sure that they are speaking a language that is understood by the company responsible for executing the testing tasks. This emphasizes the importance of having an
employee within the organization who is assigned the responsibility of monitoring the external testing process and communicating with the contractor. Allowing feedback from the people performing the tests will make necessary fine-tuning of the testing process possible. When attention is paid to some or all considerations mentioned in this section, outsourcing has the potential of releasing valuable resources that can be spent on making other parts of the web system better and more attractive.

2.5.3 The Pros and Cons of Outsourcing Web Testing

As is usually the case with most things, outsourcing web testing has its set of advantages and drawbacks. [Sis02] points out that choosing to contract out coding in addition to testing, might potentially distance project decision makers from code experience. This could cause problems at a later stage when updates, modifications or immediate changes are due. The case of having to implement immediate and critical changes poses an intricate dilemma. If testing is performed by an external party it would imply that their schedule determines when the changes can actually be tested, which could be highly inconvenient for the company that has developed the system.

On the other hand, outsourcing web testing activities frees resources and helps the developing organization to deal with time pressure. Certain things might require expert understanding and experience, and hiring necessary expertise compensates for the company not being able to acquire relevant skills within a short period of time. Having an objective third party should also lead to unbiased data, and the authority that such an actor carries makes it easier for the management to accept critical results rather than dissatisfaction coming from an internal testing department. Finally, outsourcing testing could expose the code to new testing methods and tools, thus increasing the probability of previously undiscovered errors being captured.
Categorization and Interpretations of Quality Attributes

“When one admits that nothing is certain one must, I think, also admit that some things are much more nearly certain than others.”
Bertrand Russell

CHAPTER 3:
Categorization and Interpretations of Quality Attributes

In this chapter we will try to explain how the quality attributes of our choice relate to existing quality models, ISO 9126 in particular. The intention is to map performance, reliability and robustness to a formal context to create an idea as to what purpose each of them serves, and what significance they have for a properly working system. The goal of this chapter is to show how some of the attributes relate to each other, and motivate our choice to group some of the attributes together, something that has influenced the structure of the subsequent chapters.

ISO/IEC 9126 operates with six quality attributes of software. Each of these six properties has its own sub-characteristics that help identify that particular quality attribute. The six attributes – functionality, reliability, usability, efficiency, maintainability and portability – are presented in figure 3.1.

Figure 3.1
Software quality attributes in ISO 9126

3.1 Mapping of Quality Attributes to ISO 9126

ISO 9126 operates with six quality attributes of software. Each of these six properties has its own sub-characteristics that help identify that particular quality attribute. The six attributes – functionality, reliability, usability, efficiency, maintainability and portability – are presented in figure 3.1.
Categorization and Interpretations of Quality Attributes

The most straight-forward mapping is that of reliability, which is one of the six attributes mentioned in the ISO standard. But where does robustness and performance, respectively, fit in? Efficiency has two main aspects in this standard, namely time behaviour and resource behaviour; thus we find it logical to map performance to efficiency as far as ISO 9126 is concerned. The meaning of the terms time behaviour and resource behaviour is elaborated in table 3.1.

<table>
<thead>
<tr>
<th>EFFICIENCY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time behaviour</strong></td>
</tr>
<tr>
<td>“Attributes of software that bear on the response and processing times and on throughput rates in performing its function”</td>
</tr>
</tbody>
</table>

Table 3.1: Efficiency definition, ISO 9126

Figuring out where robustness belongs, however, is far from obvious. It might seem tempting and logical to consider robustness as some kind of reliability aspect, but we will explain why this is not the case in the following section of this chapter. In “Software Engineering” by Nguyen Xuan Huy [Ngu00] it is even suggested that robustness is a branch of user friendliness, which would probably map to the usability attribute in the ISO quality model. The latter quality attribute tree is depicted in figure 3.2.

As a result we did not reach a conclusion as to what ISO quality attribute should include robustness. However, there is little doubt that reliability and robustness seem closely related, as we will explain in section 3.2.1.

**3.2 Grouping of Quality Attributes**

As we mentioned earlier, efficiency seemed like a candidate attribute for both capacity and performance. Capacity and performance are without a doubt strongly linked, and articles that we found on performance testing list capacity testing as a sub-operation of the overall performance testing mission. Hence we decided to devote a separate

**Figure 3.2**
An alternative quality attribute structure
chapter to performance testing of web-based systems, where testing of capacity, load and stress are key elements.

3.2.1 Reliability versus Robustness

We consider reliability and robustness to be complementary software properties. Reliability is concerned with the relationship between a system or component and its requirements, based on the WebSys memo “What is Robustness?” by Tor Stålhane and Jianyun Zhou [SZ03]. This implies that if a system crashes when something is performed according to the specification it is a sign of unsatisfactory reliability. On the other hand, if unspecified input is fed into the application and it breaks down or ceases to operate as a result, the system might still be reliable per definition, but it is not robust. Robustness can be viewed as a mechanism to ensure system operation in spite of invalid or illogical input. It contributes to error-free operation, but it does not guarantee reliability.

A lot of articles and software literature in general have a tendency of merging reliability and robustness, usually making robustness a sub-attribute or branch of reliability without mentioning robustness explicitly. The danger in adopting this view is the fact that it might create an illusion of satisfactory robustness contributing to a more reliable system, even though this is not the case. To avoid this illusion reliability and robustness should form two separate trees instead of figuring in the same hierarchy, as shown in figure 3.3. However, we stress the fact that these thoughts and suggestions are based on our interpretation of the quality attributes and do not represent a universal truth.

![Figure 3.3](image)

Figure 3.3
Reliability defined by ISO 9126 and robustness defined by [SZ03]

One of the main reasons why we have chosen to group these two quality attributes together to form one comprehensive chapter, is based on the practicality of testing a system’s reliability and robustness simultaneously. Solely performing robustness testing – subjecting the application to random
and illogical input – is not likely to mirror realistic events. We believe it would be a lot more realistic to "sow" illogical values and data among expected input, thus testing robustness and reliability at the same time instead of viewing them as two unrelated tasks that ought to be performed at different times.
CHAPTER 4:  
Reliability and Robustness Testing

This is the first of two chapters that deal with testing of our chosen quality attributes. It consists of two comprehensive sub-chapters, where the opening one looks at reliability testing while the last section deals with robustness testing.

4.1 Reliability Testing

This section deals with reliability testing of web applications. First we will consider different interpretations of the reliability term and state a few goals and challenges of reliability testing in a web context. Then we take an in-depth look at two testing strategies and how these approaches address reliability issues.

4.1.2 Reliability Definitions

As is the case with most software quality attributes, different institutions, organizations and software experts have their own way of defining software reliability. One of the most renowned standards in the software industry is the ISO 9126. The ISO standard refers to reliability as "the capability of the software product to maintain its level of performance under stated conditions for a stated period of time". ISO goes on to split reliability into three sub-attributes:

- Maturity
- Fault tolerance
- Recoverability

Maturity deals with software attributes that have an impact on the failure rate caused by faults in the software application. Fault tolerance focuses on the system’s ability to maintain a specific level of performance when faults or intervention in the program interface occur. Finally, recoverability refers to attributes of software that affect the system’s ability to restore both its performance level and data or information influenced by a failure. From a mathematical viewpoint reliability is defined as the probability that no error will occur in a given time interval: Reliability = P (no error in [ 0 , t > ).

ISO 9126 is, however, not the only available source for reliability interpretations. A number of articles that we dealt with during this project, have their own comprehension of the term. [Ngu00] defines reliability as "the probability that this system fulfills a function, determined by the specifications, for a specified number of input trials under specified input
Reliability and Robustness Testing

conditions in a specified time interval, assuming that hardware and input are free of errors”. He claims that reliability is derived from correctness and availability. According to [QA00] reliability of a software product is determined by its ability to “exhibit a reasonable consistency in results obtained”. They believe that the definition of “reliable” is highly subjective in a web context and that the users of an application will be the ones drawing the limits and deciding what the term’s content should be. The latter underlines the importance of capturing information from target audiences before testing activities take place.

We have chosen to concentrate on the IEEE definition. IEEE refers to reliability as "the ability of the system or component to perform its required functions under stated conditions for a specified period of time". Even though it resembles the ISO 9126 definition we feel that the IEEE version stresses the relationship between reliability and a system’s required functionality more distinctively than ISO 9126.

4.1.2 Reliability Goals and Challenges

The internet is available at every hour of every day, and an application failure is destined to have consequences in terms unsatisfied customers and potential lost orders. In the case of fatal failures there are also costs related to getting the system up and running again, preferably as quickly as possible. Producing reliable software applications does not only revolve around making sure that system failures occur at infrequent intervals. According to a Microsoft publication titled “Reliability” [MS04a], ensuring that correct results are generated, in addition to proper handling of error detection and recovery, are just as important issues of failure avoidance.

The same source claims that in order to achieve the level of reliability deemed necessary, one has to look beyond the technology itself, and emphasize the significance of involving competent personnel and a suitable development process that highlights the importance of reliability throughout the production cycle. Such a focus, combined with proper testing, should minimize the risk of software failures occurring while the system is operational and available to users.

4.1.3 Reliability Testing of Web Applications

Since system components are related and linked to each other, a component failure will most likely also affect the reliability of other parts of the application. One of the main reasons why failures occur is insufficient testing. Testing any system with respect to reliability, no matter if we are considering a web-based system or a traditional application, seeks to uncover possible failures and eliminate their causes before the final version of the system is released. However, it would be unrealistic to expect that every single possible failure, or more importantly the cause of it, can be discovered – that is simply the nature of software in general. As a result, emphasis ought to be put on testing the most likely, the most frequently
Reliability and Robustness Testing

occurring user scenarios to get a confirmation that the system functions as expected.

In this section we have paid attention to a testing strategy developed by Microsoft that seems easy to grasp and should provide sufficient coverage. [MS04a] mentions four types of testing that, when performed sequentially, help testers assess and improve the reliability of their web applications. The four testing disciplines are shown in figure 4.1

![Figure 4.1: The four steps of Microsoft’s reliability testing strategy](image)

The first two steps focus on stress testing. The intention of this is to simulate a considerable workload with the purpose of assessing how the system behaves under extreme conditions. Workloads are determined by the number of simultaneous users and what services they use. We will now go through each of the steps in turn.

4.1.3.1 Component Stress Testing

[MS04a] explains the purpose of performing component stress testing as isolating “constituent components and services, figure out what navigational, functional, and interface methods they expose, and create a test front end that calls those methods”. In other words, the test seeks to expose the chosen component to stress, beyond the estimated threshold of the system, for a certain amount of time. We will talk about stress testing in depth in the following chapter that deals with performance testing.

4.1.3.2 Integration Stress Testing

Integration stress testing implies performing stress testing on the entire system and its associated services once all chosen components have been subjected to individual stress testing. Basically, we are talking about a standard integration test, as described in chapter 2, but with stress testing motives. The goal of this step is to test the interaction between system components and verify that communication between the application and its underlying services, processes and data structures is error-free.
The integration testing will initially consist of elementary functional testing, where possible paths and user scenarios will be tested. [MS04a] also stresses the fact that test scripts should coincide with how developers and testers anticipate that the web application will be used. This implies that predicted system usage ought to be determined in advance, including what parts of the system are most likely to be subjected to considerable stress and thus might turn into a critical reliability issue. Testing resources are bound to be limited; hence it is pivotal that these resources are spent wisely and that testing efforts reflect the expected usage pattern of the web-based system. Although reliability testing will take place before the final system is released and becomes available to potential users, it makes sense to stretch the testing process as much as possible in order to discover how the system behaves in a longer time frame.

4.1.3.3 Real-world Testing

As the name suggests, real-world testing tries to make sure that the system functions properly in its intended environment. Real-world testing does not restrict itself to verifying that the application runs smoothly on its operating platform, but includes checking that the system and concurrent applications and services function flawlessly together. For this step in particular it is essential that a thorough test plan devotes a section to running and testing the system in its target environment, or if this is not feasible, an environment that closely mirrors the real, intended setting.

4.1.3.4 Random Destruction Testing

The fourth and final step of Microsoft’s proposed reliability testing strategy is random destructive testing, which challenges reliability by feeding random input to the system. The intention is to reveal whether or not the application will crash or hang as a result of illogical, unformatted input. Since this type of testing has little or nothing to do with how the actual system is intended to function, it qualifies as a suitable candidate for automated testing.

What [MS04a] basically refers to as random destructive testing, strongly resembles what we consider a branch of robustness testing, as was mentioned in chapter 3. In our opinion, and according to our definition of the robustness term, the majority of illogical, random and unformatted input will most likely affect the robustness of the system, not the reliability. Then again, this relies solely on the requirement specification contents for the specific product.

4.1.3.5 Quality Assurance Testing

In addition to the four steps just described, [MS04a] refers to a quality assurance testing process that has some kind of “affinity” to reliability. The execution of such a process should assess whether or not the system

a) has implemented the features listed in the requirement specification correctly
b) satisfies error-free operation of “popular” user scenarios

c) has a reliability profile that fulfills the initial requirements

The process of determining that reliability is improved as a result of error-free operation, as mentioned in b), assumes that input data is specified in the requirement specification, based on our interpretation of reliability. If the system appears to be unsatisfactory with regards to any of the listed points, the application will need further work to reach the desired levels of reliability and overall quality.

4.1.4 An Alternative Strategy for Reliability Testing

We believed it would be both intriguing and rewarding to elaborate on an alternative way of testing reliability in web applications – a strategy that encompasses more of an untraditional approach and thus maybe complements the four-step strategy just described. The following approach is based on an article by Tian, Ma, Li and Koru at the Southern Methodist University in Dallas, titled “A Hierarchical Strategy for Testing Web-based Applications and Ensuring Their Reliability” [TMLK03], that presents a three-tiered hierarchical testing strategy.

4.1.4.1 Strategy Foundation and Definitions

The basis for this strategy is the employment of statistical testing techniques. The purpose is not to replace traditional software testing techniques with statistical ones, but rather apply them to frequently used components and functionality. Statistical testing has the ability to simplify the priority-assignment of testing efforts based on usage rates and scenarios among existing functions and navigation patterns. This is one element that will contribute significantly to application reliability.

Another key element is the use of product reliability goals as a more objective criteria for when to stop testing, while most testing techniques cling to coverage as the preferable yardstick. This change of criteria, however, assumes that testing is performed in an environment that closely mirrors the system’s probable customer usage and target environment. If not, there is a risk that reliability assessment will end up being unrealistic.

[TMLK03] present their own definitions of two vital terms. They define reliability as “the probability of failure free-operation”, and refer to web failures as “the inability to obtain or deliver information, such as documents or computational results, requested by web users”. The majority of traditional testing techniques are capable of analyzing this type of failure. While usability focus and user training are means of conquering user-related problems and errors, web source or content failures are closely tied to services and functionality that web systems strive to provide. This is the main reason why [TMLK03] believes detection and handling of web source failures hold the key to improved reliability.
4.1.4.2 Usage Models and Key Components

Usage models play a significant part in this testing strategy. Usage models are created based on user scenarios and frequencies, as building blocks for construction, selection and execution of test cases. The article focuses on two specific usage models.

The flat operational profile, referred to as flat OP in [TMLK03], highlights frequently used operations in a list or a tree structure, along with their respective occurrence probabilities. The other usage model mentioned is the Markov usage model. This model attempts to present operational units in a Markov chain. The state transition probabilities are said to be history-independent. The probability for the entire chain can be calculated as the product of the individual transition probabilities. Markov models are best suited for software systems that consist of different stages or steps that are user-initiated and hence visible to the users. Many software applications, and web-based systems in particular, possess such features.

So what are the key components of this hierarchical strategy? [TMLK03] lists three issues of essence. First, developing a high-level operational profile that specifies main functions that will be supported by the web application, and their usage frequencies. Second, a unified Markov Model, referred to as UMM, is created for each high-level group of functions with the purpose of testing related operations and components. In addition, unified Markov models are capable of creating test cases covering high-level operations and the most significant low-level functions. The subsequent results can be analyzed to uncover bottlenecks and determine current levels of both performance and reliability. Third, thorough testing can be performed on critical parts that the unified Markov model pinpointed by using low-level models stemming from traditional testing procedures.

4.1.4.3 Operational Profile Construction and Log Analysis

Achieving high levels of accuracy is important as far as the operational profile is concerned. This accuracy relies heavily on the quality and quantity of extracted information; hence various sources should be found and existing tools utilized.

The first step in creating a flat operational profile is to determine a list of functions with their respective probabilities. [TMLK03] cites the use of expert opinion, surveys or actual measurements as means of retrieving necessary data. Keep in mind that plenty of existing products and system documentations may well work as expert opinions, while a survey is likely to bring more objective and precise information to the surface with respect to application use. The downside is the tedious and meticulous planning required to construct a successful survey. However, nothing beats actual measurements when it comes to precise verification of usage scenarios and associated probabilities. The accuracy has a price though, since the latter alternative is destined to be the most expensive one. A more affordable
approach to constructing operational profiles would be to use existing logs and system records.

Log analysis is used to produce two reports: A top access report – TAR – and a call pair report – CPR. Both reports and their properties are summarized in table 4.1. The top access report contains a list of frequently used services or web pages and their respective access counts – that is how many times users or customers have accessed a particular service or visited that specific web site. Because a lot of the individual services can be regarded as stand-alone in web systems, TAR is significant. Each unit of service mentioned in the top access report "may correspond to multiple pages grouped together instead of a single page" [TMLK03]. So although the results provide an overview of usage frequencies for each web page, they do not reveal anything related to navigation patterns.

The call pair report lists call pairs and their corresponding rate of occurrence. Call pairs describe a transition from one distinct service to another. The role of a call pair report is tying separate services together and feeding the unified Markov models with basic state transitions and associated probabilities.

<table>
<thead>
<tr>
<th><strong>Top Access Report – TAR</strong></th>
<th><strong>Call Pair Report – CPR</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>List of frequently used services and web pages</td>
<td>List of call pairs and their frequencies</td>
</tr>
<tr>
<td>Access counts</td>
<td>Basic state transitions</td>
</tr>
<tr>
<td>Units of service may correspond to several pages grouped together</td>
<td>Transition probabilities</td>
</tr>
</tbody>
</table>

Table 4.1: *Reports based on log analysis*

The two reports, combined with expert opinion, such as product information and system documentation, and a proper analysis of the survey results, will establish a solid foundation for unified Markov models construction.

### 4.1.4.4 Strategy Implementation and Reliability Assessment

On the top level of the hierarchical implementation the top access report can be used as an operational profile for statistical usage-based testing – a flat list of accessed services in decreasing order.

The call pair report can be used to group TAR entries together and then have a unified Markov model being created for each of these groups. These unified Markov models will shape middle level usage models. The bottom level of the hierarchy will consist of lower level unified Markov models, along with functions and web components deemed critical. This will include parts of the application that are frequently visited.
To sum it up, the strategy proposed by [TMLK03] emphasizes testing of functions and services that are frequently accessed at the top level, while testing routine navigation patterns and usage scenarios at the middle level. Functional areas and services that play a vital part of the web system in its entirety are then covered at the bottom level.

![Figure 4.2](Hierarchical implementation)

General reliability models and analysis techniques are supposed to provide an objective evaluation of a product’s current reliability level, thus indicating when to wrap up testing activities. These models also contribute to identification of critical areas or components that demand additional reliability attention and improvement. These wide-ranging analysis models and their features have been integrated with the hierarchical approach described in this section to support two vital reliability issues:

- Bottleneck identification and reliability extrapolation
- Reliability composition and improvement

Bottleneck identification is made on the basis of failures and component reliability being associated with particular entries in the operational profile or nodes and links in the unified Markov models. As far as the second point is concerned, varying configurations and designs are explored in order to maximize overall system reliability.

4.1.4.5 Results

According to [TMLK03] analysis results have verified that there is a significant variation in usage frequencies, thus confirming the apparent need for statistical testing and the potential value of focusing on features of high risk and frequent use.

A concrete example referred to in the same article claims that performing statistical web testing over a period of 26 days would reduce the number of defects by roughly 70%, which should affect reliability accordingly. A related follow-up project was performed, incorporating more significant variations in workload measures and extending the time frame to one year.
The resulting analysis supported the trend from the initial tests, thus increasing the validity of the results.

4.2 Robustness testing

We now turn our attention to robustness in web applications. We start by looking at definitions of the robustness term and explain briefly why testing of software robustness in web-based systems is important. We then look at our chosen definition of robustness and how the different aspects of this definition can be tested, as well as what specific issues need consideration.

4.2.1 Robustness Definitions

During this project we have come across many definitions of software robustness. The majority of these definitions overlap to a certain extent, while others complement each other. What everyone seems to agree on is that the main goal of robustness is to reduce the impact of operational mistakes, erroneous input data and hardware errors. [Ngu00] states that the main effort should be applied to frequent errors resulting from typing errors or erroneous commands, and that less likely errors can be handled more laxly as long as this will not lead to serious consequences.

“Robustness Testing – Theory and Practice” by Codenomicon Ltd. [Cod04] views the importance of software robustness from a different perspective by claiming that a substantial part of information security issues that businesses face nowadays are linked to insufficient levels of robustness. They go on to say that “all robustness problems can be exploited by causing denial-of-service conditions by feeding the vulnerable component with maliciously formatted input”.

We have, however, opted for the robustness definition of the WebSys project at the Norwegian University of Science and Technology, and will use this definition as a basis for how we believe robustness in web-based systems can be tested. The WebSys definition will be highlighted in section 4.2.3 of this chapter along with testing suggestions and challenges for each part of the definition.

4.2.2 Why Test Software Robustness?

From a technical point of view, robustness problems have a tendency of “escaping” testing activities unless it is decided to test the system’s robustness explicitly. The cause of this probably lies in the nature of robustness – that it deals with exceptional input and user scenarios as opposed to normal and straightforward operations [Cod04].

Proper robustness testing will make web applications more fault-tolerant and thus contribute to an overall feeling of quality and reliability. As we have mentioned before, robustness weighs extra heavily in a web context.
Reliability and Robustness Testing

because of the user versatility on the internet. Error-free operation is likely
to result in a satisfied user and thus increase the chances of that particular
user or customer coming back later.

4.2.3 Robustness Testing of Web Applications

The WebSys project, from this point on referred to as [SZ03], defines
software robustness as "the degree to which a system or component can
function correctly
- in the presence of invalid input or stressful environmental conditions
- on a wide range of browsers"

Figure not available in pdf

Figure 4.3
The relationship between reliability and robustness.
Processable input represents input that will lead
to meaningful operations or actions.

The definition is divided into three different types of tolerance. They
characterize software robustness as a system’s or component’s
- error tolerance
- stress tolerance
- platform tolerance

For these three terms to carry any meaning we need a definition of
tolerance. According to [SZ03] "a system or component is tolerant to an
abnormal situation if it reacts as follows:
- It gives an understandable error message
- The user is transferred to a state where he can continue operations
- All input given before this state is intact"

What an understandable error message should consist of will depend on
subjective assessment and the targeted user groups. In our opinion an
understandable error message should inform the user what user-initiated
action or operation caused the error, what the exact error is and suggestions,
if any, as to what the user can do to deal with the problem.

The claim that the user should be transferred to a state where he or she can
continue operations also raises some questions. We have asked ourselves if
it is reasonable to expect that a user would install a new web browser in
order to use a specific application, and if it is reasonable to demand that users enter some kind of queue system before they continue operation when the web application is dealing with heavy traffic and stressful conditions. As far as browsers are concerned we believe it is practically infeasible for a system to support all types and versions of browsers available. A multitude of browsers are developed by a multitude of vendors, and each of these browsers come in different versions for various operating systems. As we will bring up later on in this chapter when looking at platform tolerance, both minor and more prominent differences exist among these browsers. However, most browsers support standards created by the World Wide Web Consortium (www.w3.org); hence it is reasonable to assume that web applications that utilize techniques defined in the standards just mentioned would prove to be robust on browsers adhering to these standards. Such a statement presupposes that both web systems and browsers are implemented correctly according to the standard. As a result, testing of platform tolerance could theoretically be restricted to browsers supporting the W3 Consortium standards.

We will now go through the definition of each of the tolerance types in turn and elaborate on our interpretation of them, the testing challenges we foresee and how we believe such testing can possibly be performed to achieve overall robustness.

4.2.3.1 Error Tolerance

Error tolerance is defined by [SZ03] as “the ability of a system or component to continue normal operation despite the presence of erroneous input”. The term “erroneous input” needs a definition to ensure an unambiguous interpretation: “Erroneous input is all input that can not be used by a system or component to perform its intended function(s).”

We consider all interaction between user and the system or component as input. This interaction between user and system can take on many forms, varying from filling out text boxes with appropriate information, the simple hyperlink clicking, to moving the cursor across images that are connected to scripts, that in turn will execute when the cursor points to a certain image. The interaction processing can take place on the user’s web browser, a web server or even a server operating behind the web server, such as a database server processing incoming information from a web form.

According to [Bei90] functional testing should in theory detect all possible errors, but it would take an infinite amount of time to accomplish and hence it is not practically possible to achieve. As a result functional testing requires that a chosen limit has to be agreed upon with regards to testing time. How and where to set this testing limit will be a subjective assessment, based on the level of error-free operation deemed necessary for the application at hand. The latter is likely to be influenced by the application domain. We can perform functional testing on parts or components of the web system. We could, for instance, perform functional testing on the entire system from an end user’s point of view through a web browser, or restrict
the functional testing to subsystems or individual components, where the defined interfaces serve as a basis for testing these. This mirrors structural testing, since we would be testing the internal structures of the web-based system. With time being a crucial factor, the reliability testing strategy described in chapter 4.1.4 could prove helpful, proposing what usage scenarios and components of the system would require or benefit the most from thorough testing.

Before starting any functional testing, suitable input data should be created. When opting for functional testing to test robustness we have to make sure the data or information to be fed into the system is appropriate for this purpose. We want to test how erroneous input affects the web application, and this can only be accomplished by making sure our test data have the desired properties. What is considered erroneous input will probably vary among applications and will to a certain extent be determined by the domain. Negative numbers, extremely huge numbers, zero or non-existent dates are all typical instances of erroneous input. For textual processing an empty string or an exceptionally long one serve as candidate inputs. Certain types of data have their own evaluation rules, such as the KID number found on giros and the 11 digit Norwegian personal number.

One issue to pay attention to is the possibility of dependencies between the application’s data variables. Let us look at a web shop selling t-shirts as an example. It is destined to be a dependency between what t-shirt is chosen and what size the customer can choose. All possible combinations might not be available. Whether or not a value is erroneous could be determined by other volatile data. In our example the combination of t-shirts and sizes might depend on the stock status of the various t-shirts.

Performing functional testing manually would be a time-consuming task requiring considerable effort; thus there is no doubt that an automated testing tool would make this process a lot more efficient. This is one of the reasons why we have chosen to execute, evaluate and familiarize ourselves with two selected and currently available automated testing tools for functional testing of web-based systems.

4.2.3.2 Stress Tolerance

The next type of tolerance mentioned in the WebSys project is stress tolerance. Stress tolerance is defined by [SZ03] as “the ability of a system or component to continue normal operation despite the presence of load beyond one or more defined capacity limits”. In order to link stress tolerance with stress testing we find it suitable to quote the book “Testing Applications on the Web, Second Edition”, written by Nguyen, Johnson and Hacket [NJH03], on their definition and explanation of the latter term: “Stress testing evaluates the behavior of systems that are pushed beyond their specified operational limits (this may be well beyond the requirements); it evaluates responses to bursts of peak activity that exceed system limitations. Determining whether a system crashes and, if it does,
Reliability and Robustness Testing

whether it recovers gracefully from such conditions is a primary goal of stress testing."

This elaboration of the stress testing term matches the definition of stress tolerance well. By putting an emphasis on the stress testing definition we interpret the WebSys goal of continuing normal operation as merely a demand for the service to stay alive. Requirements and demands with respect to response time, something that will undoubtedly affect how the user experiences the system, are seemingly neglected. We can not see how continuing normal operation can imply that response times will be kept reasonably constant even during periods of significant traffic boost. Rather, response times are destined to increase when the amount of traffic hits maximum.

Traffic generators will be used during the execution of the test to simulate simultaneous users accessing the system. That way, the tester will strive to reveal application weaknesses, with an emphasis on weaknesses that lead to system crashes, in order for these and the overall robustness to be improved. The tester will also seek to verify whether or not the web system is capable of resuming normal operation by itself having suffered from a crash.

4.2.3.3 Platform Tolerance

The last type of tolerance referred to in the WebSys Memo is platform tolerance. Platform tolerance is defined as “the ability of a system to run without trouble on a wide range of browsers implemented on a wide range of hardware and operating systems”. This definition seems to be general and somewhat ambiguous. What exactly does “without trouble” mean, and what is precisely meant by “wide range of browsers implemented on a wide range of hardware and operating systems”? In our opinion such statements will have to be subjectively assessed based on the actual context. We interpret the definition to claim that the system should be robust on the platform(s) specified in advance for that particular system, with regards to error tolerance and to the degree of which the platform affects stress tolerance. Another aspect to consider is how heterogeneous the users’ client machines are, since this will strongly influence the decision of what platforms the tests will be performed on. The internet user will most likely represent the most heterogeneous user group, with a multitude of platforms present. In such a scenario, what is considered adequate to test will be a matter of judgement. One possible option is to use updated statistics on internet users’ browsers and operating systems as a basis for appraising what platforms would incorporate the user majority. Once the decision has been made as to what platforms to support, the testing can take place.

Each of the platforms will be tested to see if they satisfy the error tolerance and possibly stress tolerance requirements, to the extent a client’s platform affects server performance. In section 4.2.3.1, we discussed a possible way for error tolerance to be tested, and following this procedure for each and every one of the platforms represents a feasible solution to determine whether the platform is robust or not. However, we question if it is strictly
Reliability and Robustness Testing

necessary to execute the procedure in its entirety. While the operating environment is diverse on the client side, the server side is basically the same. Hence it is a plausible thought to consider limiting testing activities to the parts of the system that run on the client side, thereby neglecting server side components. This ought to reduce both cost and time spent on testing duties.

The various software layers can be viewed as components, where client side functionality, such as scripts, CSS and dynamic HTML, is viewed as one component and the server processing as other system components. This implies that component testing can be performed on the browser to discover errors that are not related to data or information exchange between the browser and the web server. Integration testing, on the other hand, will be capable of detecting errors that occur in the interaction between client and server.

So far we have made the assumption that only the web browser affects the robustness of the different platforms. The browser can influence parameters sent to the server, for example when a script error leads to non-existent values in a form. The browser is controlled by the data and pages that it receives from the web server. This holds even though various web browsers might be affected in different ways by the same data and pages because of differences in implementation and settings. The differences detected by various web browsers may vary from minor and insignificant ones to a level of difference that is likely to affect the usability and utility of the web application as a whole. To counteract this issue additional scripts and tags are added that perform the exact same operations, but on different browsers. As a result, a web page will contain several versions of itself, but only one of these page versions will be displayed in the current browser, which in turn will adapt and adjust the page to its desired format and execute the scripts tailored for that specific browser.

The line of action taken in the previous paragraph does not represent the only way to format pages correctly on different browsers. Server side scripting provides another alternative. When the web browser requests a file from the web server the browser transfers a parameter known as HTTP_USER_AGENT, that contains information regarding the client’s operating system, hardware and browser. This information can be used by a script running on the server to execute different parts of the script depending on the specific browser that contacted the web server.

The noteworthy advantage of server side scripts is the increased control of script execution. The user may have deactivated client side scripting for security purposes. As the example below serves to illustrate, the programming necessary for displaying a certain page the exact same way on different web browsers can be complicated and result in over-complex code that poses a threat to understandability. Although understandability is not directly linked to testing it is nonetheless a technique that is used in quality assurance work.
The example referred to above is taken from Apple Computer’s “Plug-in Detection” [App04] and shows that even though web browsers have the same manufacturer and the same version number, there is no guarantee that they will speak the same language. If a programmer was to create a procedure to verify what plug-ins were installed on a Mac version and a Windows version of Microsoft Internet Explorer, he or she would have to create both a JavaScript and a VBScript. In addition, it would be necessary to hide the VBScript within a JavaScript to make sure the user would not receive error messages from the Mac version when detecting the VBScript.
CHAPTER 5: Performance Testing

This entire chapter is devoted to performance testing of web-based systems. It opens with a general introduction to performance testing that describes primary goals and purposes. Then attention is directed to three broad testing techniques and the concept of workload, before we dive into load and stress testing processes respectively. Finally, the chapter is rounded off by looking at some testing considerations and how user load impacts three common performance measures during analysis.

5.1 Goals and Basics of Performance Testing

Performance testing of software in general seeks to identify possible bottlenecks and their causes, in addition to optimizing and tuning the platform configuration. It is a testing discipline aimed at verifying an application’s ability to operate normally under expected load levels as well as peak load conditions. Determining how well a system scales to enable increased capacity is also an issue. An article titled “Testing .NET Application Performance” by Microsoft Corporation [MS04b] claims that performance testing is about assessing how a system responds to a specified set of conditions and input, and that multiple individual performance test scenarios are required to cover all relevant conditions and input.

Performance has been a driving force in system architecture for quite a while. Since application performance greatly depends on the architecture, testing should begin once the initial architecture is in place. Performing testing during the development period enables software developers to control how new functionality affects overall performance. Although software performance is considered vital, “Software Architecture in Practice, Second Edition” by Len Bass, Paul Clements and Rick Kazman [BCK03] claims that “as the price/performance ratio of hardware plummets and the cost of developing software rises, other qualities have emerged as important competitors to performance”.

5.1.1 Performance Measures and Output

Performance testing provides us with plenty of answers regarding the tested system, with the most significant ones listed below [MS04b].
Performance Testing

- Response time
- Throughput
- The maximum amount of concurrent users supported
- Resource utilization with respect to CPU, RAM, network/disk I/O
- Behaviour under various workload patterns
- General application weaknesses
- System breaking point – the point where the application stops responding to requests

[NJH03] stresses the need to define an acceptable response time for all parts of a web-based system. Knowing that services provided by the application might have varying processing needs, they claim that different parts are likely to have different requirements. The response time will stay practically constant as long as available resources outweigh the number of users to serve, but as the competition for resources gets fiercer the response time is destined to rise. If the number of users accessing the system continues to grow, the system will eventually stop reacting, with no further requests being processed. The response time measure will be described more thoroughly in section 5.2.4.

5.1.2 Primary Performance Factors

The elements that impact system performance are many. The user’s network connection might be a restrictive factor when communicating with the web system, and the geographical location of the server and the users is likely to have some effect on performance. The infrastructure of internet suppliers has the possibility of impacting data transmission rates, thus becoming a performance factor, along with the specifications of the users’ client machines. Server configuration is yet another decisive performance factor for any web-based system. In other words, the overall performance of an application is affected by all components contributing to the execution of a service, and hence attempting to specify what factors seem feasible to test is a recommendable practice.

Since performance fluctuates in step with variations in usage pattern, performance tests ought to be repeated at appropriate intervals.

5.2 Performance Testing Techniques

According to [MS04b] performance testing consists of three major types of tests or testing techniques – load testing, stress testing and capacity testing. We will now take a brief look at each of them. Table 5.1 presents an overview of the test types and their main objectives.

5.2.1 Load Testing

A load test has the purpose of verifying system behaviour under both normal and peak load conditions [MS04b]. The amount of traffic is kept at a pre-defined level and an evaluation of the application’s performance is
carried out to determine whether an acceptable level has been reached or not. Load tests can be performed with regards to volume as well as longevity [NJH03]. Load and capacity testing are relatively similar as far as execution is concerned, and hence they are frequently mixed up. It is important to stress, though, that contrary to a stress test, a load test does not push the system beyond its breaking point. Capacity and load tests simulate regular user activity.

### 5.2.2 Stress Testing

Stress testing is performed to evaluate the behaviour of a web application when it is subjected to loads beyond its operational limits. In other words, the purpose is to see how the system handles an amount of traffic beyond what it is designed for. A stress test will reveal whether or not a system survives sudden, short-lived peaks in traffic load, crashes, and in the case of a crash – if the system is capable of restoring normal operation. The motivation is to reveal the weak parts of the system – the elements that caused the break-down – and consequently improve these.

### 5.2.3 Capacity Testing

Capacity testing complements load testing by determining the particular server’s failure point. Load testing, on the other hand, monitors results at different levels of load and traffic patterns. Capacity testing enables identification of a scaling strategy to assess whether to upgrade the current server(s) or employ additional servers, also known as scaling out. In [NJH03] this type of testing is referred to as performance testing, where the amount of traffic exposed to the system is regulated and the resulting performance is measured at different traffic loads.

<table>
<thead>
<tr>
<th>Load testing</th>
<th>Stress testing</th>
<th>Capacity testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify application behaviour under normal and peak load conditions</td>
<td>Verify application behaviour when pushed beyond operational limits</td>
<td>Determine server failure point</td>
</tr>
<tr>
<td>Enables measurements of performance variables</td>
<td>Uncover weak “links”</td>
<td>Identify scaling strategy</td>
</tr>
</tbody>
</table>

Table 5.1: Goals of load, stress and capacity testing

### 5.2.4 Definition of Key Terms and Measures

What all three testing types have in common is that the web system will be subjected to traffic and information will be retrieved and analyzed to predict at which levels of traffic the application will be overpowered. They are
performed to confirm that requirements with respect to acceptable user experience are satisfied.

User experience is measured in two dimensions – correctness and timeliness. The former deals with the system functioning correctly even at high loads of traffic, while timeliness is defined based on response time. The shorter the system’s response time is, the better the user experience is likely to be, because shorter response times lead to reduced waiting time for the user. Acceptable user experience can thus be defined as correct processing of user request, with a resulting response time that is shorter or equal to the user’s expected response time. A user’s expected response time is tied to two factors, according to [NJH03]: “The value of the product and service that business offers, and the customer’s value system expectation”.

To be even more specific we should mention that response time is defined as the time it takes from a user clicks on a link and sends a request, until results can be observed.

Performance objectives are related to four measures or performance terms in particular [MS04b]:

- ✔ Response time
- ✔ Throughput
- ✔ Resource utilization
- ✔ Workload

Response time, also referred to as latency, is the time elapsed until a request has been processed. Response times can be measured on both server and client, where the latter includes the request queue, network latency, as well as the time required by the server to complete request execution. Throughput defines the number of requests that the web system is capable of serving per unit time. Requests per second is the most common measure of unit as far as throughput is concerned. Resource utilization with respect to CPU, RAM, disk I/O and network I/O, represents a cost in system operation. Resource cost can be computed per operation, and is usually measured for a certain user load or distributed on the basis of a workload profile.

Workload revolves around simultaneous and concurrent users. Simultaneous users have active connections to the same web site, whereas concurrent users are accessing the site at the exact same time. A workload profile, thus, consists of a likely user composition where the users perform various system operations. Simulating simultaneous users can be done by making sure the test scripts incorporate so-called think time. The purpose of think time is to ensure that not all user requests being simulated will occur at the same time. Removing think time from the test script makes sense if the goal is to stress test the web application by simulating concurrent users.
5.3 Workload

Workload plays a vital part in performance testing. The performance of a web application is inevitably affected by the number of simultaneous users and the services they access. In order to attain a reasonably accurate assessment of system performance, the number of users and their usage patterns – illustrating what functionality is being used by the various users – should resemble the eventual reality facing the system. The feasibility of determining the total number of users will depend on whether the system is an internal business application residing on the company’s intranet, or if it is located on the internet. Estimating the number of users and what services they will employ is considerably easier when dealing with an intranet than with the internet. Once the system is operating in its real environment we will be able to uncover differences between expected and actual user activity.

It is reasonable to assume that different user groups will make use of different services and different parts of the functionality offered, and hence subjecting the web system to various types of load. This is something that should be taken into consideration during performance testing, by simulating the various user groups working against the same system.

5.3.1 Workload Modelling

Workload modelling implies identifying one or several workload profiles. These profiles represent variations with respect to key scenarios, the number of simultaneous users, the request rate and request patterns, and will in turn be simulated on the system to be tested. A workload model describes how each key scenario is performed, in addition to identifying user types and significant characteristics. [MS04b] lists several important questions aimed at application workload assessment. We will now go through each of the concerns that these questions address.

5.3.1.1 Key Scenarios

The system-critical scenarios, seen from a performance viewpoint, need to be identified. These scenarios ought to be uncovered early during the development phase, preferably during requirements specification.

5.3.1.2 Maximum User Amount Expected

A web application’s maximum operational capacity is determined by the number of simultaneous users – the number of users that have active connections to the same web site [MS04b]. The operational capacity of a system needs to be addressed to help determine the workload.
5.3.1.3 Possible User Actions

Considering what set of user actions are possible is important. Logging in and out, browsing product catalogues and placing orders are a subset of typical user actions.

5.3.1.4 Existing User Profiles

The existing user profiles for the web application can be categorized based on user types and their associated user actions. Thus, they create a platform for classifying user profiles.

5.3.1.5 User Profile Operations

In addition to classifying user profiles, the operations performed by an average user for each of the user profiles need to be identified. Such identification can be based on marketing data for new web applications, or alternatively on log analysis of web servers for an existing system.

5.3.1.6 Average Think Time

As we mentioned earlier in this chapter think time represents the time a user spends between two consecutive requests, for instance when reading web page information or filling out a form. According to [MS04b] think time can be averaged out for all requests.

5.3.1.7 Expected User Profile Composition

In order to determine the percentage mix of business actions executed by users, the usage pattern for each key scenario of the application should be used as a guideline.

5.3.1.8 Test Duration

The final issue that needs to be addressed with regards to the system workload, is the duration for each of the tests to be performed. An e-commerce application with a reasonably stable user profile will require substantially shorter test duration times than web sites experiencing greater variations in user profile on a particular day. In certain cases it might even prove necessary to run tests continuously for up to several weeks in order to evaluate how the system behaves in the long run.

5.4 Performing Load Testing

The purpose of load testing, as was mentioned in section 5.2.1, is to verify a web system’s behaviour under normal and peak load conditions, with the load increasing incrementally to observe application performance [MS04b]. Performing load testing renders possible bottleneck identification and determining the system’s operational capacity.
The load testing strategy presented in this chapter is based on a six-step approach proposed by Microsoft, and recommends performance goals, system characteristics, workload characteristics as well as test plans as input to the process. The process blueprint is shown in figure 5.1 below, and contains the same steps as the stress testing strategy described in the subsequent section.

![Figure 5.1 Process blueprint for load and stress testing of web applications](image)

### 5.4.1 Step 1 – Identify Key Scenarios

The first step aims at discovering scenarios requiring specific performance goals or scenarios having substantial influence on the overall performance of the application. A scenario may play a key role because of frequent execution or because of considerable resource demands.

### 5.4.2 Step 2 – Identify Workload

In this step the focus rests on determining three performance attributes for each scenario identified in the previous step:

- ✓ User amount
- ✓ Request rates
- ✓ Request patterns
The total amount of users includes both simultaneous and concurrent users. Request patterns help to determine average user load and the request rate for particular functions or services of the web system. According to [MS04b] the load testing process should begin by distributing users with reference to the particular user profile, assuming that a workload model has already been constructed. The load should increase incrementally for each test cycle.

5.4.3 Step 3 – Identify Metrics

Selecting appropriate metrics requires assessing what is relevant with regards to performance goals and the bottlenecks to be identified. Metrics are typically categorized. The various metric types and their associated descriptions are summarized in table 5.2.

<table>
<thead>
<tr>
<th><strong>Metric category</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Network-specific metrics</td>
<td>Network efficiency</td>
</tr>
<tr>
<td>System-related metrics</td>
<td>Server resource utilization</td>
</tr>
<tr>
<td>Platform-specific metrics</td>
<td>Software running on destination platform</td>
</tr>
<tr>
<td>Application-specific metrics</td>
<td>Inserted code to monitor system health and uncover performance problems</td>
</tr>
<tr>
<td>Service level metrics</td>
<td>Application throughput and latency</td>
</tr>
</tbody>
</table>

Table 5.2: Metric classification

Once the metrics have been assessed, the next step is to determine acceptable load levels for the system.

5.4.4 Step 4 – Create Test Cases

Based on the workload patterns resulting from step 2 of the load testing process, all test cases should be documented in tests plans and users should be divided into user profiles. Expected results can also be listed, if deemed necessary or useful.

5.4.5 Step 5 – Simulate Load

Automated testing tools are preferably utilized to execute the key scenarios and to simulate load. [MS04b] stresses the importance of not overloading the client machines used to generate load, since this is likely to affect performance measures.

5.4.6 Step 6 – Analyze Results

In the final step of the process the output data produced in step 5 will be analyzed and compared to the metric levels assessed in step 3. Data will be
analyzed based on the performance goals that were referred to as process input at the start of chapter 5.4:

✓ Throughput versus user load
✓ Response time versus user load
✓ Resource utilization versus user load

How user load affects throughput, response time and resource utilization respectively, is graphically indicated in section 5.7 of this chapter – Analysis of Performance Data.

If multiple test iterations are performed, the load should be increased incrementally until the threshold limit of the application is surpassed. Studying the metrics data should give an idea as to the possible bottlenecks. Once this has been accomplished, modifications should be made to the system and the tests need to be run once more.

5.4.7 Benefits of Load Testing

This load testing process produces updated test plans, identifies the behaviour of the system under varying levels of load, helps to determine the maximum operating capacity of the application, as well as pointing out potential bottlenecks and recommendations as to what might cause them.

5.5 Performing Stress Testing

Having presented a way of performing load testing of a web-based system, we now turn our attention to stress testing. To brush up on what was written in section 5.2.2, stress testing involves subjecting the application to substantial or even extreme loads that are beyond the system’s capacity, while at the same time denying the application the required resources to be able to deal with the load [MS04b]. The objective is to uncover possible bugs that only manifest themselves at high load levels. This includes issues such as synchronization problems, race conditions – meaning that the output of a process or function exhibits unexpected, critical dependence on relative timing of two system events, memory leaks, and loss of data as a result of network congestion.

The stress testing strategy described in this section is closely related to the load testing process just elaborated, following the same 6 steps of figure 5.1. The first thing that needs consideration is the identification of one or possibly more scenarios that need to be subjected to stress testing. The level of detail can vary greatly as far as stress testing is concerned – from testing a web page to dealing with separate classes, functions or methods. The process requires that application characteristics, likely scenario issues, workload profile and the peak load capacity resulting from the execution of load testing, are all in place [MS04b].
5.5.1 Step 1 – Identify Key Scenarios

The first step involves selecting one or multiple scenarios that are considered relevant to stress test. If more than one scenario is identified, the criticality of each of them should be evaluated, based on how influential they are with respect to overall system performance. Logically, it is recommendable to focus testing efforts on operations and user actions that are likely to impact the performance significantly. The load test will probably give valuable input to this step, having identified areas that greatly affect performance.

5.5.2 Step 2 – Identify Workload

The load applied to the chosen scenario(s) is meant to stress the system beyond the threshold set for the web application. As we have mentioned earlier, load can be increased incrementally to observe the resulting behaviour under various levels of load. There is, however, an alternative path available – the creation of an anti-profile to the workload model initially developed during load testing. Constructing an anti-profile implies inverting the workload distributions for the scenario to be tested. The load that is left is distributed among the remaining scenarios. The load should be increased for the scenario under consideration while evaluating system response at different levels of load. For more details on the anti-profile approach we refer to [MS04b].

5.5.3 Step 3 – Identify Metrics

In this step, the objective is to decide, for each of the scenarios, which metrics correspond to possible scenario pitfalls, with the metrics being related to throughput and performance goals. We will bring up the topic of metrics again at the end of this chapter.

5.5.4 Step 4 – Create Test Cases

Once the metrics are successfully identified, the test cases are supposed to be documented in test plans according to the workload patterns assessed in the second step of the stress testing process. As was the case with the corresponding step of the load testing procedure, expected results can be listed if desirable.

5.5.5 Step 5 – Simulate Load

Step 5 also resembles the corresponding step of load testing. Automated testing tools are used to execute the scenario(s) and simulate load. The same considerations need to be taken with respect to client machines when performing stress testing, as was the case with load testing.
5.5.6 Step 6 – Analyze Results

The final step comprises analyzing the data captured and comparing them to the defined acceptance levels of the identified metrics. If the resulting analysis was to claim that the desired or required performance level has not been reached, prospective bottlenecks and their associated causes need to be dealt with. In order to address bottleneck issues it might prove necessary to carry out a design review or code review.

5.6 An Alternative Outlook on Performance Testing

With reference to [NJH03], performance testing can be characterized as a capability planning process, and should not be seen as a test resulting in a “passed” or a “failed”. The performance testing process is divided into three phases – planning, testing and analysis.

5.6.1 Planning Phase

During the planning phase the system, its workload and acceptable response times for the various parts of the application, are defined. Additionally, one should clarify what metrics will be collected for future analysis. Planning also embraces the creation of a plan for what testing techniques will be employed and when they will be used. These things are documented in a test plan. Test scripts that seek to emulate users are also created. When the analysis phase is reached, the gathered metrics are analyzed in order to reveal potential bottlenecks. Improvement suggestions can be prepared and new tests will be designed. The data to be used during testing can be made manually or be generated by a script. Alternatively, data stored in an existing system can be reproduced.

5.6.2 Testing Phase

During the actual testing there are two primary ways of generating system traffic. People can be used to produce necessary traffic, but this approach is highly unpractical, especially for systems that need to be subjected to considerable amounts of traffic. The other option is to make use of an automated testing tool, such as the ones we have tested briefly in chapter 6. Testing tools can either be purchased or developed in-house. A third alternative is to use an application service provider – ASP – which generates traffic from several locations around the world.

5.6.3 Analysis Phase

During the analysis phase it is recommended that all data is organized and characterized. Subsequently, the data is analyzed with the purpose of laying
open potential sources of performance issues. Web-based systems are typically built on several hardware and software systems, and performance problems in any of these might cause poor performance for the entire system. The results are documented in a test report.

5.7 Guidelines for Successful Performance Testing

There are plenty of things to be done that will contribute to meaningful performance testing, just as there are numerous issues to avoid. We have chosen to group the considerations to be made into three broad categories that will be treated in turn.

5.7.1 Environmental Considerations

For obvious reason one should strive to make sure the system is running on a hardware configuration that resembles the end environment as much as possible, as well as assessing the correct server software. Having pinpointed this, it should be emphasized that tests should by no means be executed in live production environments, primarily because of possible network traffic. Instead, stick to a representative and isolated test environment. Another thing to consider is making sure that performance testing is done without the use of local caching or buffering, in order to capture realistic response time measures.

[MS04b] recommends the use of a single graphical user interface client to capture end user response time during load generation. As we have briefly mentioned in step 5 of both the load and stress testing processes, the client machines should not be overly stressed. The reason is simply that excessive processor and memory usage may contribute to making the client machine a performance bottleneck. All machines operating during the test, including the load-generating client machine, must be monitored.

5.7.2 Simultaneous and Concurrent User Considerations

[MS04b] stresses the importance of differentiating between simultaneous and concurrent users. To brush up on what was written in the introductory parts of this chapter, simultaneous users have active connections to the same web site, while concurrent users connect to a web site at the exact same time.

Performance testing by simulating simultaneous users has the advantage of rendering possible an assessment of the actual load that the web system is capable of handling without resulting in a flood of errors of the type “server busy”. To simulate concurrent users, think time can be removed from the test script. This is ideal for subjecting a system to stress testing, where all simulated users access the web site concurrently.
5.7.3 Miscellaneous Considerations

We have described the concept of think time earlier in this chapter, reflecting the elapsed time when a user is being inactive with regards to his or hers interaction with the system. While eliminating think time can be suitable in a stress testing sense, removing think time between to subsequent requests is highly unlikely to indicate a realistic event. Thus, one should keep in mind that think time adds realism to a performance test, and that it is destined to vary depending on the content of the page being viewed. Using different user parameters will also contribute to a more realistic load simulation.

Scenarios should be ranked according to the criticality of the functionality they employ. Also, make sure that the test load script simulates a suitable load for each scenario, without generating redundant requests [MS04b]. And finally, never try to provoke system break-downs while performing a load test. A load test should ideally focus on practical considerations.

5.8 Analysis of Performance Data

Performance analysis is closely related to the chosen metrics. However, what metrics to capture will vary depending on what type of server we are dealing with. There are significant differences between a web server and a database server, for instance, and thus there are server-specific metrics in addition to metrics that are of interest on all types of servers. The latter metrics are typically tied to resource utilization.

![Throughput versus user load](image)

**Figure 5.2**

*Throughput versus user load*

As we mentioned in step 6 of performing load testing, the captured data needs to be analyzed and compared against the acceptance levels assessed for each relevant metric. Figure 5.2 illustrates how throughput typically is
affected by user load. The point on the curve identified as point-of-service failure represents the maximum user load that the web system can manage while still meeting service level requirements with regards to requests per second.

Figure 5.3
Response time versus user load

Figure 5.3 above depicts the general relationship between response time and user load. The response time stays reasonably linear from low to medium levels of load, before exploding as a certain limit is reached. The arrow points to the amount of simultaneous users the web application can handle while still meeting service level goals with respect to scenario response time.

Figure 5.4
Processor utilization versus user load

Figure 5.4
Processor utilization versus user load
Finally, figure 5.4 illustrates how user load typically impacts processor utilization, measured in percentage. The threshold limit indicates the workload for the application that is within the specified limit for processor utilization.
"Experience is that marvellous thing that enables you to recognize a mistake when you make it again."
Franklin P. Jones

CHAPTER 6:
Evaluation of Automated Testing Tools for Web Applications

In this chapter we will evaluate two automated testing tools for web applications, namely Mercury QuickTest Professional and Segue SilkTest. These testing tools aim at assisting software testers when performing functional testing. Functional testing can be said to be a set of tests that seek to verify that a system functions properly – that the system can perform tasks that it is set out to do. Tests can be created to evaluate whether or not a system is capable of executing operations identified in the requirement specification, or as we will do: Find out how the system copes with unexpected, random input.

6.1 Evaluation Criteria

We had little or no knowledge of software testing prior to this project. As a consequence we had to anticipate what software-assisted testing could do for us. We made a list of evaluation criteria that we believed would help us assess important features of the tools. The list contained the following criteria:

- Time spent on learning the product
- Coverage
- Type of errors detected
- Test set generation
- Reporting and presentation of test results
- Time consumption
- Multi-platform support

This list was based on the knowledge we accumulated during the project and the expectations we had built up concerning the software tools. It was slightly modified to compensate for elements initially neglected as the evaluation got under way.

6.2 Target Web Application

An authorization was received to test a survey-generating web application, residing on the intranet pages of the Norwegian University of Science and Technology. Testing will be restricted to the registration part of the system, where information such as title, description of the current survey and miscellaneous properties for the survey need to be filled out. Additionally, questions including their associated
answering alternatives and question properties must be registered, as shown in figure 6.1.

Figure 6.1
Survey construction

6.3 Installation and Learnability

We did not see the point in using quantitative measures for determining how much time was spent on learning the products. Since we will never be fully comfortable with all features of such comprehensive tools and thus continually gain system-specific knowledge
the more time we spend on each of them, we have tried to use qualitative measures instead.

6.3.1 Mercury QuickTest Professional 8.0

Once the installation was complete, a tutorial was displayed. Mercury had developed their own web application for booking of flight tickets that could be tested in the tutorial. We went through the process step by step in order to grasp how the software could be used to test the web application. This made it easy to get started and enabled us to get familiar with the basic functionality while performing tutorial examples.

To get to know the tools and their functionality without following the tutorial recipes, we chose to develop a very simple web page that requested two numbers from the user and then calculated the sum. Once we started to use QuickTest to test this web page, we realized that tool usage was not as straightforward as the tutorial suggested.

6.3.2 Segue SilkTest 7.1

Installing Segue SilkTest was more time-consuming and cumbersome than what was the case with Mercury QuickTest. In addition to downloading the program itself we had to download and install a “helper application” with the purpose of monitoring our license to make sure it was valid at the time. Several files had to be downloaded individually as well.

Like Mercury, Segue came with a comprehensive tutorial to enable the user to get comfortable with the software, along with a sample web application for testing purposes. The tutorial contained a lot of information between the different exercises that took time to digest. Hence we felt that it took more of an effort, particularly time-wise, to come to grips with the core functionality of SilkTest. Segue also had several tutorials accompanying the actual software product, and for a 14-day evaluation period it became a few too many documents to deal with. As a consequence we did not become aware of a separate tutorial that explained how to create a test that would retrieve test data from an external data source. This was unfortunately discovered only after a series of trials and errors.

Once we started to use SilkTest to test our own applications we came across a problem. The program had difficulties identifying the one part of our web page that we wanted to verify. The solution to this problem was to modify the web page code slightly. The seriousness of this issue, however, is up for discussion, since the majority of tests will probably be performed by organizations that have access to the source code of the tested application. Once this problem was out of the way we were ready to set up the tests to verify the functionality of our target web application. The verification
process had been presented in the tutorial, but even so it took us some time to map the tutorial’s process description to our particular system. Eventually we succeeded in constructing a test for our simple summation application.

6.4 Testing of a Web-based System for Performing Surveys with Mercury QuickTest

While initially QuickTest seemed easier to use than SilkTest, it did not prevent us from encountering numerous problems in testing our target web application. With the limited time as an additional stress factor it turned into a task that challenged our patience. The web application that we were provided with consists of several pages in HTML format. To start the creation of a test we let QuickTest register our user actions when performing a survey registration in Microsoft Internet Explorer. The resulting test will be referred to as a basic test and serves as a starting point for constructing a series of tests that receive their data from QuickTest’s built-in spreadsheet. Thus, the test will consist of several iterations of the basic test, with different data for each iteration.

![Survey overview](image)

**Figure 6.2**
Survey overview

6.4.1 Element Identification

Once the creation of a basic test is completed, QuickTest builds a database consisting of all web pages, including the elements residing on these pages. The elements, known as HTML tags, are identified by combinations of tag...
type, the text enclosed within tags, and possibly properties and property values that they contain. If several elements exist that can not be unambiguously identified with the identification procedure just mentioned, an index is used in addition, with the purpose of separating these particular elements. To enable flawless operation of QuickTest it is essential that web pages do not change. If changes are inevitable actions should be implemented to contribute to QuickTest’s understanding of how to deal with changes. The importance of this particular aspect was something we got to experience when working with our target application.

A section of our target web application is shown in figure 6.2 above. The registered surveys are listed, where two of them have the exact same name. To differentiate between the two link elements QuickTest will utilize the index referred to in the preceding paragraph. The testing tool will generate a script based on the manual survey registration performed in our web browser, and when we click on the link “Forhåndsvurdering” found in figure 6.2, the following line will be added to the script:

```javascript
Browser("Browser").Page("Spørreundersøkelse_13").Link("Forhåndsvurdering").Click
```

![Dialog box for element identification](Figure 6.3)

The link will present us with a preview of the surveys, but exactly what survey will we get to see? The survey overview shown in figure 6.2 lists three different surveys with three associated links, all of which says “Forhåndsvurdering”. QuickTest hides parts of the information that it uses to
identify the element of use, thus compelling us to open a dialog box in order to find element properties. This dialog box is shown in figure 6.3, and tells us that link number two is chosen, since the index starts at 0.

Having given an introduction to how QuickTest uses available values and characteristics to navigate through web pages, we feel it is appropriate to explain one of our encountered problems and how it was solved.

At the time we registered our basic test there was only one registered survey in the database; hence the action of clicking the link “Forhåndsvisning” had no need for an index value, being the solitary link carrying that particular text on the web page. We modified our basic test to contain a series of tests that read test data from QuickTest’s integrated spreadsheet. This new test went through the process of registering several surveys, and at the completion of each of them the test was intended to preview the current survey and verify the correctness of data. However, this turned out to be problematic since QuickTest kept previewing the first survey and tried to control these data against data registered during the test. This goes to show how sensitive automated testing is to changes in web pages. In this case the issue was solved by deleting the survey as the last point of action in each iteration, after the survey had been verified. Consequently, only one survey was registered in the database at any time, resulting in only one link carrying the text “Forhåndsvisning”.

6.4.2 Verification Through Checkpoints

Being a software application for automated functional testing, QuickTest seeks to verify input data, namely textual information and actions, against system feedback. Our initial intention was to create tests to demonstrate instances of unsatisfactory robustness in the target application, but this posed too many challenges and worries for both testing tools and hence we decided to leave the idea. The use of HTML as input data has the potential to modify the web page, making it practically impossible for QuickTest to verify it. QuickTest performs verification based on what Mercury refers to as checkpoints. Different types of checkpoints test separate parts of a web page. In addition to web page testing, checkpoints are used to test the correctness of database queries and whether or not a web page is easily understandable for functionally disabled users. We merely used checkpoints for verification of text and HTML elements.

Verifying text means that all text within an HTML formatting is grouped together in order to be verified as a text string. If the text is composed of several variables or results from a computation, the entire text element has to be tested against the expected text specified by the tester. We tried to combine input data with constant text values to create strings to be verified against web page data, without success. In spite of this QuickTest might well support such tasks. One good thing about QuickTest was the presence of a checkpoint type that made it possible to detect a variable between two constant text strings, thus saving us the trouble of specifying expected data, assuming we knew the particular variable in advance. Unfortunately we did
not succeed in using this checkpoint type when several variable data existed or when the variable was not “enclosed” on both sides by static text strings. Parameter values in an HTML page can also be verified by a checkpoint type, for example to determine if a button is active or has been deactivated.

A noteworthy finesse regarding QuickTest is the integrated web browser that visually depicts in what part of the web application the user is when working with the test script. This browser was also useful when inserting checkpoints into the test.

6.5 Testing of a Web-based System for Performing Surveys with Segue SilkTest

To start testing with Segue SilkTest we began by creating an empty project. Username and password are required in order to gain access to the university's intranet, and hence this action had to be registered in SilkTest by defining two constants in a configuration file. We feel it could have been desirable to have some sort of protection for this type of information. Setting up a test project in SilkTest consists of more steps than what was the case with QuickTest. We especially noticed the sequence of terminating the browser, Microsoft Internet Explorer in this case, only to relaunch it right afterwards. Once the project was fully configured we were ready to record a survey registration, which would make for a basic test.

![Figure 6.4](image)

*Dialog box for specifying element values in SilkTest*
6.5.1 Creating a Basic Test

To create our basic test we put SilkTest in recording mode and performed the survey registration process in our target web application. Having completed the registration the next step is to preview the survey and verify the correctness of the registered data. Points of verification are set during the recording phase by pointing the cursor at the element to be checked and then pressing Ctrl-Alt. This triggers a dialog box to pop up, shown in figure 6.4, where the user gets to specify what parts of the element should be controlled and against what values verification will be done. The current element values are set as default.

Once we were done with the recording process we executed the test, only to encounter an issue related to data verification. There was a mismatch between the information we had initially registered in the form during the recording process and the test input data. It turned out that language-specific letters, such as æ, ø and å, were treated differently during registration than during the actual test. Once we fixed this problem everything went smoothly.

6.5.2 Constructing a Data-driven Test

Having successfully created a basic test we turned our attention to data-driven tests. Creating a data-driven test implies “transforming” our basic test to a test consisting of multiple iterations, where each iteration fetches its own data from a data source, such as an Excel spreadsheet or an Access database. In order to create such a test we closely followed SilkTest’s Data-Driven Workflow and used an Excel spreadsheet containing various data as our data source. Once the data sources were properly specified, we searched through the test and replaced all input data in the test script with code that enabled data retrieval from the Excel spreadsheet(s). Figure 6.5 illustrates the task of replacing values.
6.5.3 Element Identification

The basic test was created similarly to the one made in QuickTest; hence the intention was to delete the survey after it had been verified by the testing tool to avoid the existence of several identical links – “Forhåndsvisting”. SilkTest would always choose the same link that was used during recording unless it was told otherwise. At the time we created the basic test there were no other surveys registered in the database, and consequently SilkTest would choose the survey at the top of the list. The technique that had worked flawlessly in QuickTest did not attain the same success this time. When SilkTest detects an error, the application terminates the script and moves on to the subsequent iteration without deleting the current survey.

To get around this problem we changed our strategy and focused on how SilkTest makes use of an index to address the correct element in the tested web page. While QuickTest uses an index to differentiate between similar elements, SilkTest uses the index as a means of numbering elements of the same type. To ensure that the correct link was chosen, we declared an index value for each test iteration. Since these index values could be affected by input data, they become input data themselves that need additional specification. Creating input data can thus become a challenging prospect, seeing that changing data might cause various index values to be modified. We put the index value in a separate spreadsheet column, and after a while we accomplished accessing this newly created column from the SilkTest environment. We stress the importance of making sure that the starting point of the test remains the same for each test run. For instance, if one or more surveys from the previous run are still registered, all index values will be displaced, possibly resulting in error occurrences. Numerous times we had to terminate test execution because we had forgotten to delete surveys stemming from previous test runs.

During the execution of data-driven tests we kept getting error messages from SilkTest, claiming that the test set contained empty fields. Unfortunately we did not find a suitable solution to this problem, but opted to fill the “holes” with data in order to progress. However, creating sterling and adequate tests for bigger applications demand technique and experience that can not be attained within a two-week evaluation period. We learned a lot, but still feel we only scratched the surface of such a comprehensive testing tool.

6.6 Type of Errors Detected

Both testing tools use web browser response in their mission of error detection. As a result it is unlikely that the tool will be able to discover errors that will not be reported back to the user. Both Mercury and Segue elaborate on methods for testing databases in order to control correct storing of data, in the help files accompanying the software products. Other underlying services are not as standardized and hence more problematic to test.
At the start of the evaluation we were thinking of using the tools to test certain aspects of robustness. However, we believe it is not likely that automated functional testing can be employed to automate the robustness testing process. The main reason for this statement is that general robustness testing will require substantial subjective judgement, and it would seem unrealistic for testing tools to be capable of assessing the appropriateness and understandability of an error message. Nevertheless automated testing tools can be used to find suitable tests, that in turn will be manually examined to assess robustness.

6.7 Coverage

A test does not necessarily have to prove that the entire application is correct. Input data controls the execution of the program and hence what portions of the program code will be tested. For instance, the code for web shop payment might contain different program code for VISA and MasterCard payment. Test coverage is a measure, usually expressed as a percentage, for determining how big a portion of the code has been tested. Prior to inspecting the testing tools we had hopes of them being able to estimate the level of coverage for each executed test. At the end of the evaluation period, however, we know that this feature is not supported, suggesting that either we look at other products or calculate the figures ourselves.

Knowing that test coverage is determined by input data and user actions, neither SilkTest, nor QuickTest should have any influence on this, unless they skip certain input data or actions.

6.8 Test Set Generation

A sub-goal of our initial plan was to test our target web application with respect to robustness, but with no requirement specification present it comes down to pure speculation to separate reliability from robustness, with reference to chapter 3.

We have tried to get an overview of what possibilities SilkTest and QuickTest have for generating test data. The former contained functionality for generating random numbers and text strings, while QuickTest merely supported random number generation. When asked about this issue Mercury recommended us to generate test data through script creation. With the limited time available we did not see the point in putting extensive effort into exploring additional unknown terrain. Instead we opted for a Microsoft Excel spreadsheet for generating input data, creating tables of data and making use of a formula to randomly select table data for test set generation.
6.9 Reporting and Presentation of Test Results

In a separate result file QuickTest presents a table with all test iterations and their associated results – passed or failed. In addition, a test summary is displayed below the iteration results, reporting the number of passings, failures and warnings encountered during the test run.

![Figure 6.6: Presentation of test results in QuickTest](image)

As figure 6.6 suggests, QuickTest uses a tree structure to represent the iterations and their associated test data. By navigating in this tree structure we can easily go to the location where an error occurred and get detailed information about it. The standardized reporting of errors can, if considered feasible by the tester, be supplemented by adding separate messages during test execution. The report is saved in XML format.
Whereas QuickTest reported whether or not each iteration had passed the test, SilkTest looks at the test as a whole, in addition to reporting the percentage of test cases that have passed and failed, as well as the number of errors and warnings. Like QuickTest, a tree structure provides a means of getting information on the causes of a failed iteration. Numerous reports can be generated based on the result set, with several sorting criteria available. The presentation of test results is shown in figure 6.7 below.

Figure 6.7
Presentation of test results in SilkTest

6.10 Time Consumption

With our lack of experience with and knowledge of the software tools it became a time-consuming task to complete an automated functional test. Since it turned out to be a cumbersome process to get everything right, we shrank our test down to 10 iterations of the basic test to save some time. Even with substantial testing experience and
thorough tool knowledge, setting up a test properly takes time. Once this task is completed the reward can eventually be cashed in – a considerable amount of tests that can be executed based on the setup work.

### 6.11 Multi-platform Support

QuickTest supports the latest versions of the Microsoft Windows operating system, including Windows 2000, XP and 2003 Server, along with Microsoft Internet Explorer, Netscape and AOL web browsers. SilkTest supports the same operating systems and web browsers, and even reports supporting older versions of Microsoft Windows, such as Windows 95 and 98.

From a platform tolerance perspective it is commendable that different browser versions and operating systems from various vendors are supported. According to browser statistics published by Refsnes Data [RD04] over 90% of internet users have a variant of Microsoft Windows, while just under 6% used Linux or Mac OS in October 2004. Since we are viewing testing of platform tolerance as a series of error tolerance tests on the different platforms we would want to support, it would have been desirable for testing tools such as SilkTest and QuickTest to support even more platforms and browsers. We have not paid attention to user groups that utilize hardware such as mobile phones or Portable Digital Assistants for web application use. The differences between these groups and general PC users are probably more significant than the variation between different configurations of operating systems and web browsers on PCs.

Testing tools capable of making automated testing on different platforms and web browsers possible, would have simplified the testing of platform tolerance considerably. Thus, we would have preferred to see both QuickTest and SilkTest and their respective tests being executable on additional platforms and browsers. The reasons why the software producers restrict their support to Microsoft Windows only, will simply lead to speculations, but it is likely to assume that the distribution of alternative operating systems and web browsers is a vital factor. As long as the dominating vendors’ competing forces are relatively many and individually too small to make an impact on the market, it is probably not an easy decision to make as to which further vendors to support. Another issue to consider is whether or not the current number of non-Windows users is sufficiently high to make the creation of separate robustness tests for these user groups seem desirable and economically justifiable. This is, however, not a matter we intend to discuss in this report.

In SilkTest’s “Getting Started” tutorial Segue claims that a test can be created with a web browser and executed in different web browsers, such as Netscape, AOL and Internet Explorer, and various versions of web browsers with minor modifications. This, in addition to how they elaborate on tests being capable of running on several distributed machines in a network, gets us dreaming about how these features can combine to allow for multiple
platforms and web browsers to be tested simultaneously, thus increasing the efficiency of platform testing.

6.12 Summary

One of the first lessons we learned was that there is no such thing as a shortcut to well-functioning tests. Automated testing tools, such as SilkTest and QuickTest, and the quality of the tests produced rely on the people creating the tests and the test data used. When we started the evaluation we aimed at evaluating the tools with respect to error tolerance and robustness, but in posterity we realized that the angle we chose would hardly have mattered. Regardless of what type of functional testing we would have opted for, the software tools would have been employed similarly, knowing that the choice of test sets is the decisive factor as to what we actually test.

6.12.1 Tool Similarities

Both testing tools were capable of registering our actions in the web browser by generated code to recreate the scenarios formed. SilkTest displayed the registered code while QuickTest let us choose between a code representation and a graphical tree structure representation of our actions. The respective interfaces for test representation are presented in figure 6.8 and 6.9.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>No measures of test coverage</td>
</tr>
<tr>
<td>Types of errors detected</td>
<td>To enable error detection, data has to be exposed in order for the testing tools to verify them</td>
</tr>
<tr>
<td>Test set generation</td>
<td>Generation of test sets not a primary function in QuickTest nor SilkTest</td>
</tr>
</tbody>
</table>

Table 6.1: Summary of tool similarities

Another significant similarity is the use of an index. Both QuickTest and SilkTest made use of an extra index to distinguish between HTML elements that would otherwise be impossible to separate. Since automated testing tools in general are not equipped with “common sense” we got to experience how difficult web page orientation and correct navigation can be. However, control over produced code simplifies testing by labelling the parts of the code to be used with unique identifiers. Some additional tool similarities are listed in table 6.1.

6.12.2 Tool Differences

QuickTest allows the user to make test modifications through dialog boxes, while SilkTest requires that changes be done in the generated test code. The use of dialog boxes limits what operations the user can perform, thus contributing to swift and easy operation without requiring considerable
experience and knowledge. Without a thorough understanding of how SilkTest and its script language function, the process of having to write code manually turned out to be rather cumbersome. Having said that, using QuickTest means that the user will be entering and exiting dialog boxes rather frequently; hence we believe that developing tests by programming will be the more effective alternative in the long run.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Mercury QuickTest</th>
<th>Segue SilkTest</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learnability</strong></td>
<td>- Well written tutorial&lt;br&gt;- Easy to get familiar with basic functionality</td>
<td>- Comprehensive collection of tutorials&lt;br&gt;- Overwhelming information</td>
</tr>
<tr>
<td><strong>Test representation</strong></td>
<td>- Code&lt;br&gt;- Tree structure representation</td>
<td>- Code</td>
</tr>
<tr>
<td><strong>Element identification</strong></td>
<td>- HTML tag type&lt;br&gt;- Enclosed text&lt;br&gt;- Tag properties&lt;br&gt;- Property values&lt;br&gt;- Index value to separate identical elements</td>
<td>- HTML tag type&lt;br&gt;- Enclosed text&lt;br&gt;- Tag properties&lt;br&gt;- Property values&lt;br&gt;- Index value for indexing elements of equal tag types</td>
</tr>
<tr>
<td><strong>Verification</strong></td>
<td>- Verification added after test recording</td>
<td>- Verification added during test recording</td>
</tr>
<tr>
<td><strong>Presentation of test results</strong></td>
<td>- Passes and fails of each test iteration&lt;br&gt;- Test summary&lt;br&gt;- Hierarchical tree structure</td>
<td>- Test summary&lt;br&gt;- Hierarchical tree structure</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td>- Integrated spreadsheet&lt;br&gt;- Integrated browser</td>
<td>- External data sources</td>
</tr>
</tbody>
</table>

Table 6.2: Summary of important tool characteristics

The testing tools also differ from an interface perspective. QuickTest reminds us of a typical Microsoft Office application based on its use of
dialog boxes and point-and-click functionality, while SilkTest resembles an integrated development environment, consisting of a category tree where project files are grouped together. For instance, all files containing test scripts were placed in a separate “Script” folder, as shown in figure 6.8.

QuickTest’s integrated web browser was a feature we highly appreciated, since it enabled us to easily add checkpoints by pointing and clicking on elements in the browser. SilkTest, on the other hand, made us add verification points during test recording. If we forgot to add a verification point we could either manually add the piece of code ourselves or record the verification and paste the resulting code into its proper place. Knowing that test discipline can not be learned in a flash, along with the limited time contributing to somewhat unstructured testing, the adding of checkpoints and verification points was necessary a couple of times.
Figure 6.9
QuickTest's graphical test representation, along with the built-in spreadsheet and browser

6.12.3 Final Comments

We do not feel that a two-week evaluation period has allowed us to gain the knowledge necessary for pinpointing weaknesses with regards to the two products. Such an assessment would be unfair based on our lack of experience. On the other hand, it has been an instructive process that has enabled us to get a valuable glimpse of how HTML resources are utilized to make automated testing possible.
"A conclusion is the place where you got tired of thinking"
Arthur Bloch

CHAPTER 7: Conclusion

In the introductory chapter, composed at the start of the project period, we set forth a number of project goals that were meant to drive the assignment work. Thus, we have used our stated objectives as a platform for assessing what lessons have been learned.

7.1 Summary

Web-based systems have presented a wide range of new challenges to the software community. Greater focus on graphical user interface as a driving force in development, along with shorter and more intensive iterations, the employment of new technology and products, and the reduced development time to minimize time-to-market, are all factors that lead web applications into a category of its own. Logically, these characteristics complicate testing of such systems, hence making the testing process a suitable candidate for outsourcing. Although the benefits of outsourcing testing activities are many and valuable, there are a few concerns that need addressing before such an important decision is reached.

The heart of this report has dealt with testing of three pivotal quality attributes in the context of web-based systems – performance, reliability and robustness. We have experienced that the lack of acclaimed standards or the presence of too many standard definitions of the quality attributes, could be a primary reason as to why communication of “quality knowledge” is challenging. As a result, article contributors tend to begin by elaborating on an exhaustive definition of their concepts, before diving into the core content. Hence, we have come to realize that a lack of precise, unambiguous definitions makes generalizing knowledge from various sources a daunting task.

During this project we have learned how reliability can be tested through a four-step process, performing component stress testing, integration stress testing, real-world testing and random destruction testing in sequence, while rounding off with quality assurance testing to ensure that the desired level of reliability has been reached. We have also come to know alternative approaches, among them an hierarchical testing strategy that also enables reliability assessment. The approach focuses testing efforts on important and frequently used scenarios and navigation patterns. With the lack of available literature on robustness of web-based systems, we built or own apparatus of concepts, based on [SZ03], to address robustness issues, considerations and suggestions. Finally, we dealt with performance testing, embracing load,
stress and capacity testing. In particular, we became aware of the difference between subjecting a web application to load and stress testing respectively. Load testing focuses on verifying system behaviour under normal and peak load conditions. Stress testing, on the other hand, applies loads that are beyond the system’s capacity, while at the same time denying the application the required resources to deal with the load.

Our final goal listed in the introduction was to familiarize ourselves with automated testing tools, to get a glimpse of the functionality offered and how this could translate into a more effective testing process. The most important lesson was that there is no such thing as a shortcut to high-quality tests. The quality of the produced tests is at the mercy of the ones creating them and their corresponding test data – a software tester’s experience and skills are crucial in order for an automated testing tool to become an asset. However, current testing tools do not support a wide range of operating platforms and browsers if we choose to look beyond Microsoft products. We believe that multiple platforms might gain importance if users start to employ operating systems that exhibit a greater degree of disparity than what is the case between different Windows versions. Tools that automate testing on various platforms have the prospect of making multi-platform testing more cost-effective and improving quality on less common platforms.

7.2 Further work

While working on this project a few ideas have matured with respect to future work. In chapter 6 we referred to Segue SilkTest presenting the possibility of running tests on several distributed machines in a network. Combining these features to enable simultaneous testing of multiple platforms and web browsers, led us to envisage increased efficiency of platform testing. Hence, we see the need, although a challenging one, for developing automated testing tools that support a wider range of platforms.

It might also prove interesting to perform a comparison, possibly through a formal experiment, of executing testing manually and through the use of an automated testing tool. This could be attractive with regards to several aspects, e.g. the type of errors detected, the number of errors discovered and time consumption, to name a few. Although we acknowledge some complicating factors such as the time it takes to fully utilize the target testing tool, it still strikes us a relevant and feasible prospect.

Finally, the lack of literature on software robustness implies that there is room for further, more extensive exploration. Compared to quality attributes such as performance and reliability, robustness appears to be a rather vague concept. All in all, this leaves a lot to be desired, both with regards to robustness metrics and a formalized testing process.
Appendix A

☑ References & Bibliography

[QA00] QA Labs Inc.  
– “The Living Creature” – Testing Web Applications  
http://www.qalabs.com/resources/thelivingcreature.pdf

[Bei90] Boris Beizer  
– Software Testing Techniques, second edition  
ISBN 0-442-20672-0

[CO02] COAST Web Quality Management  
– Web Quality Testing and the Rational Unified Process  

[Sis02] Derek Sisson, philosophe.com  
– Outsourcing Web Site Testing  
http://www.philosophe.com/testing/outsourcing.html

[Dav02] Neil Davidson, Red Gate Software Ltd.  
– Web Service Testing  
http://www.red-gate.com/dotnet/more/web_services_testing.htm

[MS04a] Microsoft Corporation  
– Reliability  

[TMLK03] Tian, Ma, Li & Koru, Southern Methodist University, Dallas  
– A Hierarchical Strategy for Testing Web-Based Applications and Ensuring Their Reliability  
http://userpages.umbc.edu/~gkoru/papers/HIE-COMPSAC-03.pdf

[SZ03] Tor Stålhane, Jianyun Zhou  
– WEBSYS Memo – What is Robustness?

[NJH03] Hung Q. Nguyen, Bob Johnson, Michael Hackett  

[App04] Apple Computer, Inc.  
– Plug-in Detection  
Appendix A – References & Bibliography

[BCK03] Len Bass, Paul Clements, Rick Kazman
  – *Software Architecture in Practice, Second Edition*

[Ngu00] Nguyen Xuan Huy, National Center
  for Natural Science and Technology, Vietnam
  – *Software Engineering*
  [http://www.netnam.vn/unescocourse/se/software.htm](http://www.netnam.vn/unescocourse/se/software.htm)

[Cod04] Codenomicon Ltd., Finland
  – *Robustness Testing – Theory and Practice*

[MS04b] Microsoft Corporation
  – *Testing .NET Application Performance*

[RD04] Refsnes Data
  – *Browser Statistics*
  [http://www.w3schools.com/browsers/browsers_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp)

[Bry03] Brian Bryson, IBM Rational
  – *Bridging the Gap Between Black Box and White Box Testing*

  – *The Free Encyclopedia*
Appendix B

**Glossary**

- **Active connection**
  An ongoing connection to the web server that occupies server resources. A server will have limitations as to the number of active connections it can serve.

- **ActiveX**
  A Microsoft standard, also referred to as Component Object Model (COM), that is used for communication between software programs. ActiveX enables communication between components, such as Microsoft Internet Explorer and third-party components.

- **Black box testing**
  Also known as functional testing. Black box testing involves testing the system from a user’s perspective, exposing it to different types of input and checking whether or not the resulting output is in accordance with the specification [Bei90].

- **Bottleneck**
  A restrictive factor with respect to web system performance.

- **Checkpoint (Mercury)**
  A point in the test where accordance between web page and expected values is controlled.

- **Concurrent users**
  Users accessing a web site at the exact same time.

- **COTS**
  Commercial Off-The-Shelf. Individual software components with specialized properties that can be employed instead of developing the necessary code. COTS are typically used to reduce development costs and time-to-market.

- **CSS**
  Cascading Style Sheets. A computer language used to describe the presentation of a structured document written in HTML or XML [Wik04].

- **Database server**
  A computer or software program that offers database services to other programs or computers.

- **Data-driven test**
  A series of test runs where each test execution uses different data.
Appendix B – Glossary

✓ **Dynamic HTML**
   A technique for creating interactive web pages. Client side scripts, CSS and HTML can be combined to make a page interactive.

✓ **Error**
   Incorrect behaviour resulting from a fault [Bei90].

✓ **Failure**
   Incorrect behaviour of a component [Bei90].
   The lack of ability of a component, equipment, subsystem or system to perform its intended function as designed [Wik04].

✓ **Fatal failure**
   In our context a fatal failure is an error that causes the system to break down and thus be unable to provide the users with services.

✓ **Fault**
   Incorrect program or data object – a bug [Bei90].
   An abnormal condition or defect at the component, equipment or subsystem level, which may lead to a failure.

✓ **Fuzzy requirements**
   A description of incomplete, incorrect requirements.

✓ **HTML**
   Hypertext Markup Language. Web page format for documents.

✓ **IIS**
   Internet Information Services. Microsoft’s web server software.

✓ **Integration testing**
   Tests performed to check for inconsistencies between integrated modules or units.

✓ **Java Applet**
   An additional module that can be integrated into a web page, assuming that the client machine is running software known as Java Virtual Machine.

✓ **Learnability**
   A quality attribute referring to how easy it is to understand and learn to use an application.

✓ **Legacy system**
   In our context a legacy system is a computer system that primarily is in use because it provides the organization with an important service, and replacing it would prove costly.
✓ Markov chain
   A Markov chain is a discrete-time stochastic process with the Markov property. In such a process, the distant past is irrelevant given knowledge of the recent past [Wik04].

✓ Markov model
   A model based on Markov chains.

✓ Markov property
   A stochastic process where the conditional probability distribution of future states of the process, given the present state, depends only upon the current state [Wik04].

✓ Metric
   A measure of some property of a piece of software or its specification.

✓ Operating capacity
   A measure of how much load, with respect to the number of users and operations, a web application can handle.

✓ Platform tolerance
   Sub-characteristic of robustness, referring to how well a system functions on various platforms.

✓ Peak load
   Maximum levels of traffic, thus crossing the system’s load limit for a period of time.

✓ Portability
   The ability of a software application to perform basic modifications in order to run on different platforms.

✓ Quality attribute
   A measurable part of the system that is used to quantify its quality.

✓ Quality model
   A formalization of quality work through a specified work process for quality promotion.

✓ Real-world testing
   Testing performed in a system’s real environment or in a resembling environment.

✓ Regular software
   In our context this implies traditional software that is installed and running locally on a computer.
Appendix B – Glossary

 ✓ **Scalability**
   A quality attribute referring to how well an application handles variations in the total number of users, the number of operations per user or other variables affecting resource usage.

 ✓ **Scenario**
   A set of expected actions performed by a potential user of the application.

 ✓ **Security mechanism**
   A mechanism that is designed to detect, prevent or recover from a security attack.

 ✓ **Service Level Agreement**
   A contract between a network service provider and a customer that specifies what services will be provided.

 ✓ **Simultaneous users**
   Users that have active connections to the same web site.

 ✓ **SSL**
   Secure Socket Layer. A protocol for encryption of data to be sent through a network.

 ✓ **Structural testing**
   See White box testing.

 ✓ **System testing**
   Testing of quality attributes that are affected by the entire system, e.g. performance.

 ✓ **Test plan**
   Also known as a test design, a test plan is developed prior to implementation work and is intended to prevent errors from occurring. A test plan explains how testing should be performed.

 ✓ **Test set**
   A set of input data and expected output data used to test a system.

 ✓ **Think time**
   Time added to test scripts to make user simulations more realistic, since users are likely to spend some time on a web page, either reading information or filling out forms, before moving on.

 ✓ **Third-party products**
   Denomination used to describe additional products developed by a vendor other than the developer of the main product.

 ✓ **Throughput**
   The number of requests a web application is capable of handling.
Appendix B – Glossary

**Time-to-market**

The time it takes from product development is started to the same product is completed and available to the market.

**Tolerance**

In our context, tolerance refers to an application functioning in a specified way, in an abnormal situation – to what extent the system gives an understandable error message, keeps user input intact and transfers the user to a state where operation can continue.

**Traffic generators**

Software or hardware tools used in performance testing to simulate users of web-based systems.

**Unit testing**

Unit testing, also known as component testing, seeks to verify that the target component or unit satisfies the functional requirements, and that the structure resembles the planned structure.

**Usage model**

A model representing how the various services of an application are used and their associated frequencies. Usage models can be employed during test creation, selection and execution.

**Verification (SilkTest)**

A point in the test where accordance between web page and expected values is controlled.

**Web application**

Consists of clients and one or more physical servers performing up to several types of services. A web application can incorporate specialized servers such as a web server, database server and an application server that interact to perform necessary services for the web system.

**Web-based system**

See web application.

**Web browser**

A web browser is a software package that enables a user to display and interact with documents hosted by web servers [Wik04].

**Web server**

A computer running a software program responsible for serving web pages, mostly HTML documents, via the HTTP protocol to clients [Wik04].

**Web service**

A web service is a collection of protocols and standards used for exchanging data between applications. Software applications written
in various programming languages and running on various platforms can use web services to exchange data over computer networks like the internet [Wik04].

✓ Web system
  See web application.

✓ Web-based system
  See web application.

✓ White box testing
  Performing tests on functions that are not directly available for the users of the final product.

✓ Workload
  Represents an estimation of different user groups and their system usage. The purpose of a workload is to simulate users of the application during performance testing.

✓ XML
  Extensible Markup Language. A W3C recommendation for creating special-purpose markup languages [Wik04].
Appendix C - Index

✓ Index

Access;28; 59
Acrobat Reader;2
active connections;40; 41; 48
ActiveX;2
ad hoc testing;1
Apache;9
application domain;33
application failure;23
application servers;8
applications;1; 2; 3; 6; 7; 8; 10;
15; 23; 25; 27; 33; 54; 60
Automated testing tools;44; 46;
65
avoidance;23
basic test;55; 57; 58; 59; 60; 63
black box testing;14; 15
bottlenecks;12; 27; 37; 44; 45; 47
browser;3; 7; 9; 10; 32; 33; 35;
56; 58; 60; 64; 65; 67; 68
browser statistics;64
call pair report;28
capacity;VII; 4; 19; 33; 37; 38;
39; 41; 42; 45; 68
changes;6; 7; 8; 12; 17; 56; 57; 65
checkpoints;57; 58; 67
clients;1; 12
code;VI; 13; 15; 17; 35; 44; 47;
54; 59; 61; 65; 67
code complexity;13
Component;24
component testing;13; 35
corrupt;25; 38; 40; 44; 48
corrupt users;38; 40; 44; 48
contractor;16
COTS;2
Coverage;52; 61
crashes;20; 34; 39
CSS;35
cursor;32; 59
database;3; 8; 10; 32; 49; 55; 57;
59; 60
database servers;8
data-driven test;59
Data-Driven Workflow;59
Definitions;22; 26; 30
destruction testing;67
detection;13; 15; 23; 26; 60
developers;8; 13; 25; 37
development;2; 4; 6; 7; 8; 9; 12;
13; 16; 23; 37; 41; 67
development technologies;2
distributed;40; 46; 64; 68
efficiency;18; 19; 44; 65; 68
errorous commands;30
error tolerance;34; 64
error-free;13; 20; 24; 26; 32
errors;3; 9; 12; 13; 15; 17; 23; 26;
30; 32; 35; 48; 52; 60; 62; 63;
68
evaluation;29; 33; 38; 52; 54; 60;
61; 65; 68
evaluation criteria;52
Excel;VI; 59; 61
failure free;26
fatal failures;23
feedback;9; 17; 57
Flash;2
flat operational profile;27
framework;9
functional testing;4; 12; 14; 25;
32; 33; 52; 57; 61; 65
functionality;VI; 1; 9; 10; 12; 18;
23; 26; 35; 37; 41; 49; 54; 59;
61; 67; 68
GUI;6; 8
hardware;23; 30; 34; 35; 37; 48;
64
HTML;35; 55; 57; 65; 68
hyperlink;7; 32
IIS;9
index value;57; 60
information;3; 9; 10; 12; 22; 23;
26; 27; 28; 30; 32; 33; 35; 39;
42; 52; 54; 56; 57; 58; 59; 62;
63
input;3; 13; 14; 16; 20; 21; 22; 25;
26; 30; 31; 32; 33; 37; 43; 45;
46; 52; 57; 59; 60; 61
Appendix C – Index

installation; 7; 8; 54
Integration; 13; 24; 35
Integration testing; 13; 35
interfaces; 13; 33; 65
iteration; 9; 55; 59; 60; 62; 63
Java Applet; 2
learnability; 10
legacy system; 10
link; 33; 40; 56; 57; 60
load testing; 38; 39; 42; 43; 44; 45; 46; 49; 68
maintainability; 18
Markov usage model; 27
Mercury; 4; 52; 54; 55; 57; 60; 61
metrics; 44; 45; 46; 47; 49; 68
navigation patterns; 26; 28; 29; 67
normal operation; 32; 33; 34; 39
operating capacity; 45
operating systems; 32; 34; 64; 68
operational limits; 33; 39
outsourcing; 6; 16; 17; 67; 69
parameter; 35
peak load; 37; 38; 42; 45; 68
performance; 3; 4; 8; 9; 10; 12; 18; 19; 22; 24; 27; 34; 37; 38; 39; 40; 41; 42; 43; 44; 45; 46; 47; 48; 49; 67; 68
performance testing; 4; 19; 24; 37; 38; 39; 41; 47; 48; 67
platform; 3; 25; 31; 32; 34; 37; 42; 44; 52; 64; 65; 67; 68
platform tolerance; 32; 34
portability; 18
programming language; 2
project; 1; 2; 3; 4; 8; 12; 16; 17; 22; 29; 30; 31; 33; 52; 58; 67; 68
quality assurance; 3; 7; 16; 25; 36; 67
quality attributes; VI; 4; 10; 14; 18; 20; 22; 67; 68
quality models; 4; 18
QuickTest; 4; 52; 54; 55; 56; 57; 58; 60; 61; 62; 63; 64; 65; 66; 67; 68
Real-world testing; 25
recovery; 23
regular software; 6
reliability; 3; 4; 10; 18; 19; 20; 22; 23; 24; 25; 26; 27; 29; 30; 61; 67; 68
requirements; 6; 8; 9; 12; 13; 14; 20; 25; 26; 34; 38; 40; 41; 50; 52; 61
requirements specification; 9; 25; 26; 41; 52; 61
resource utilization; 44; 45; 49
Response time; VI; 38; 40; 45; 50
robustness; VI; 3; 4; 10; 18; 19; 20; 22; 25; 30; 31; 32; 33; 34; 35; 57; 61; 64; 65; 67; 68; 70
robustness testing; 4; 20; 22; 25; 30; 61
scalability; 12
scenarios; 24; 25; 26; 27; 29; 30; 37; 41; 43; 44; 45; 46; 65; 67
script language; 66
scripts; 9; 25; 32; 35; 40; 47
security mechanisms; 9
Segue; 4; 52; 54; 58; 60; 64; 68
Server side scripting; 35
service; 2; 10; 12; 28; 30; 34; 38; 40; 47; 50
SilkTest; VI; 4; 52; 54; 55; 58; 59; 60; 61; 63; 64; 65; 67; 68
simultaneous users; 24; 34; 40; 41; 48; 50
software engineering; 1; 2; 3; 6; 7; 9
source code; 54
spreadsheet; VI; 55; 57; 59; 60; 61; 68
SSL; 9
stress testing; VI; 12; 24; 33; 34; 37; 38; 43; 45; 46; 48; 49; 67
Stress Testing; 24; 39; 45
stress tolerance; 33; 34
Structural testing; 15
subsystems; 33
survey; 27; 28; 52; 55; 56; 57; 58; 59; 60
System testing; 13
technology; 1; 2; 6; 23; 67
test design; 13
test plan; 25; 47
Test set; 52
Test Set Generation; 61
testing databases; 60
testing strategies;22

testing techniques;1; 6; 12; 13; 26; 37; 38; 47

testing tools;2; 4; 33; 52; 57; 60; 61; 64; 65; 66; 68

think time;40; 42; 48; 49

third party products;8

throughput;19; 40; 44; 45; 46; 49

Time consumption;52
time frame;2; 8; 25; 29
time-to-market;2; 8; 67
tolerance;10; 22; 31; 32; 33; 34; 64; 65
top access report;28
total quality;16

traditional systems;1

Traffic generators;34

Transition probabilities;28

transmission rates;38
tutorial;54; 55; 64
typing errors;30

unified Markov Model;27

Unit testing;13

usability;9; 10; 18; 19; 26; 35

Usage models;27

usage pattern;25; 38; 42

usage rates;26

user interaction;2; 7

user interface;2; 6; 12; 48; 67

verification;27; 54; 57; 59; 67

version management;2

weaknesses;9; 34; 38; 68

web applications;1; 2; 4; 6; 8; 9; 10; 22; 24; 26; 30; 32; 42; 43; 52; 67

web server;2; 10; 32; 35; 49

web services;6; 10; 12

web site;9; 10; 12; 28; 40; 41; 48

web systems;1; 2; 3; 4; 6; 7; 9; 16; 26; 28; 32

Web-based Systems;1; 2

white box testing;15; 70

Workloads;24

World Wide Web;1; 32

World Wide Web Consortium;32