QUALITY ASSURANCE OF WEB APPLICATIONS: A SURVEY

WENJUN XU

A MAJOR REPORT

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

MARCH 2003

Abstract

Quality Assurance of Web Applications:

A SURVEY

Wenjun Xu

This report presents a survey on quality assurance of Web applications. Starting from the introduction of Web application architecture, it discusses the quality factors of Web applications. First, the basic software measurement concepts are introduced. A quality model specifies which properties are important to a Web application and how these properties are to be determined. Based on the specification of the entity type, the attribute definition, metric formula, and application procedures, a catalog of metrics can be used to support different quality assurance processes. Quality control and assurance are widely applied in today's Web-based system development processes. User-centered quality engineering is introduced in the context of Web application development processes, which results in increasing of software quality and end-user satisfaction. Tools and measurement methods are developed and applied in the Web applications quality assessment procedures - SUMI is a method to evaluate the software system usability and a Web-centered Function Points measure gives the estimation of the maintenance cost of Web system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Quality Needs of Web Applications

The Web is becoming an important channel for critical information and the fundamental technology for information systems of the most companies and organizations. Nowadays is widely accepted that a Web site is not merely a matter of contents and aesthetics: demanding functionality is required, combining navigation with complex services and transactions. Many users already rely on the Web for up-to-date personal, professional and business information. The substantial changes transforming the World Wide Web (WWW) from a communication and browsing infrastructure to a medium for conducting personal businesses and e-commerce are making quality of Web applications and services an increasingly critical issue. For example, users are not willing to tolerate latency times greater than eight-ten seconds [VAL01]. Some recent work [NEL00] suggests that not only must a Web site be available for customer access, page-to-page response time must not exceed four seconds, or the customer will likely exit and seek alternative sites. Also if response time becomes too high, there will be many periods of unavailability, and the security is not fully guaranteed. Thus, the inability to provide service because of a system failure is a question of reliability and robustness. This scenario motivates the need to design and implement architectures being able to guarantee the service level agreement (SLA) that will rule the relationship between users and Web applications.

Web applications in electronic commerce (e-commerce) involve sharing business information, maintaining business relationships and conducting business transactions by means of telecommunication networks [ZWA96]. The medium of Web applications is the WWW, which growth has been unprecedented, with millions of users worldwide. Consumers can use the Web to purchase all kinds of good and services, like books, cars, flowers, food, banking, entertainment, etc. It is easy to understand that e-commerce has brought a whole new range of services to all these users. For this reason, its growth has been impressively rapid. By the year 2003, the number of people purchasing goods and services online will have trebled, compared to nowadays, while it is expected that 80% of companies will connect to the Web [GHO98], meaning that they will be enabled to conduct electronic transactions.

Because of the complexity of Web infrastructure, many components could affect the quality of Web applications, from network technology and protocols, to hardware and software architectures of Web applications. Network carriers that have a full control on their backbones can provide the network availability and guarantee network response time. Well-designed application architecture can greatly benefit not only developers and maintainers but also different final users. In fact, a healthy navigational structure, an appropriate setting of services and contents, in addition to planned quality attributes are key factors for the success of a Web Application.

1.2 Web Application and Software Quality

Software quality is a complex, highly context-dependent concept. Software quality is one of the three most important software product characteristics, which are quality, cost, and schedule [PRE97]. The purpose of software quality activity is to "*identify, monitor, and control all activities, technical and managerial, which are necessary to ensure that the software achieves the specified SIL (Software Integrity Level)*" and functional performance, safety, reliability and security requirements [GIL92]. Different software quality requirements have been modeled by decomposing them into various quality factors, quality criteria and quality metrics in a hierarchical way. Examples of well-known existing quality models are the McCall's quality model [McCall 1977], Boehm's quality model [Boehm 1978], and the ISO 9126 Standard Quality Model [ISO 1991]. Fenton suggests monitoring the software quality in two different ways [FEN97]:

- *The fixed model approach*

  It is assumed that the quality factors needed to monitor a project are a sub set of those in an already published model.

- *The "define your own quality model" approach*

  In Software Quality Factor-Criteria-Metrics Framework [IEEE Std 1993], defined the technique to build a quality model according to the organization goals and management requirements. The model's flexible hierarchical structure is obtained by decomposition of every quality requirement in quality attributes (factors) from the management and user-oriented views; decomposition of each factor in software-oriented attributes (criteria) from the technical personnel views.

Quantitative representation (software metrics) of the characteristics of each criteria are identified and associated to the established criteria and factors.

In this report, we choose the second approach to represent the quality of Web applications. Because the final goal of the Web applications is the satisfaction of end users, the quality of Web applications is decomposed focusing on user-oriented views in quality attributes, etc. As the Web is playing a central role in diverse application domains such as business, education, industry, and entertainment, etc., there are increasing concerns about the ways in which Web applications are developed and the degree of quality delivered. Thus, there are compelling reasons for a systematic and disciplined use of engineering methods and tools for developing and evaluating Web sites and applications [MUR01]. To obtain reliable information about the product's quality, these methods should identify which attributes and characteristics should be used to assure specific evaluation goals given a user viewpoint.

In terms of quality characteristics as defined in the ISO 9126, we can consider an attribute as a direct or indirect measurable property of an entity (a Web application). Use a quality model to specify such characteristics, sub-characteristics and attributes. These quality, cost, or productivity requirements are often quoted as non-functional requirements in the literature. In this context, stakeholders should consider which are the characteristics and attributes that influence the product quality and quality in use. Specifically, there are some characteristics that influence product quality described in the ISO 9126 standard: *usability, functionality, reliability, efficiency, portability, and*

maintainability. Considering the importance of user factor in Web activities, the quality assessment of Web sites and applications should focus on those attributes that are perceived by the user. For instance, *Broken links*, *Orphan pages*, *Quick Access Pages*, *Table of Contents*, just to quote a few of them. By controlling quality requirements in new Web development projects and by evaluating requirements in developing phase- a user-centered design, can improve the product quality by discovering the absent features, poorly implemented requirements, design and implementation drawbacks, which related to the interface, navigation, accessibility, search mechanisms, content, reliability and performance, etc.

1.3 Overview of This Report

Web application system is client/server software that is connected by Internet technologies, the collection of hardware and software that comprises the network infrastructure between consumers and providers of information. Web application can be accessible by specific client software or by one or more related Web pages that are logically grouped for a specific productive purpose. The complex of network infrastructure affects the quality of Web application and services. As the Web applications are implemented on top of locally and geographically distributed network systems, the Web application infrastructure affects the performance of the application.

In **Chapter 2**, we give an introduction of Web architecture and the classification of Web applications.

In **Chapter 3**, we introduce basic definitions of software metrics and software measurement. Software metrics are being used by the Software Assurance Technology Center (SATC) at NASA to help improve the quality by identifying areas of the software requirements specification and code that can potentially cause errors. We address the classifications of software metrics and how software metrics are related to software engineering life cycle. Some widely used traditional and object-oriented metrics are selected and addressed in detail. This chapter is rather independent than the rest chapters in this survey.

Chapter 4 introduces the quality assurance in Web application development processes.

In **Chapter 5**, quality factors related to Web applications are discussed. Software metrics for these quality factors are summarized.

Chapter 6 introduces measurement tools in quality evaluation and testing of Web applications.

Chapter 8 includes the conclusion and research direction.

Chapter 2

Classification of Web Applications

The explosive growth of Internet has motivated deployment of a great variety of applications, ranging from simple text based application to multimedia, meta-computing services, etc. Today, with the fast evolution of Internet based technologies, many business mission critical applications have been also migrated towards the Web. The Internet -specifically the World Wide Web - has been treated seriously as a platform not only for information sharing among the mass public, but also to create useful Web Applications that provide significant value to customers. However Internet is still relatively new. We discuss certain issues before tackling the aim of Web application quality factors.

2.1 Evolution of Internet

The Internet was originally designed in the research and development in scientific and military fields for allowing computers to share information in the early 1960s. While computing developed and penetrated society in many ways over the succeeding decades, the Internet grew more slowly until its commercialization in 1995, which led to an explosion of growth that continues today.

Many people may think of World Wide Web as Internet, but WWW was not released until 1991, which was originally used for information distribution based on hypertext - a system of embedding links in text to link to other text, which now we call pages.

Commercial use was prohibited until the early 90s when independent commercial networks began to grow. Between 1980 and 1994 the growth rate was close to one hundred percent per year. Internet is still growing fast and today the growth rate is about sixty percent per year. This growth is measured in number of hosts.

The following data shows the use of Internet in the day-to-day life:

- The number of people online worldwide as of August 2001 is 513.41 million.
- ...the active number of Internet users in the United States is only 37 million, well below the widely reported range of 50 million to 70 million seen in most published report. (*Bits & Bytes*, by Michael Bush, 1998)
- ...about 15 million of the total 23 million U.S. households on the Internet receive their online services through AOL. (AOL Eyes Half Of All New Online Users, 1998)

Today, the Web reaches almost all possibly target groups [GRE02]. For example, consumers can buy books and compact discs online, people use Web to find jobs, and employers use Web to find employees; stocks are bought and sold using online applications provided by brokerages; and travelers book flight and hotel reservations through Web, to name a few. More and more new businesses merge into the Web. New

business requires new features. All this results in the growing complexity of the sites on the Web. Another fact is the increasing number of personal computers in the world meanings that Internet is now available to more or less every one. This makes the Internet even more interesting to new companies, which in addition means that it will keep growing.

2.2 Basic Definitions and Architecture

A Web Application is a client/server software system that is connected by Internet technologies to route the data it processes [GRE02]. Internet technologies here mean a collection of hardware and software that comprises the network infrastructure for the Web. Web Applications can be made accessible by specific client software or by one or more related Web pages that are logically grouped for a specific business purpose [GRE02], such as to buy a book, to process stock orders, etc. Figure 2.1 shows the basic architecture of Web applications [JIM].

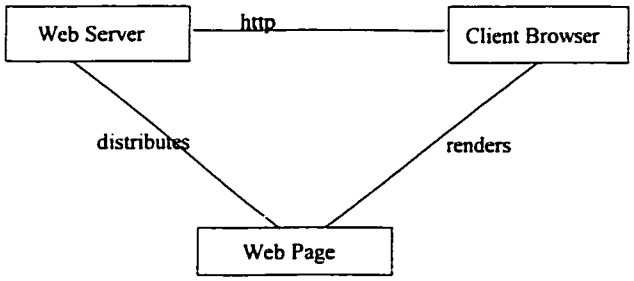

Figure 2.1 Basic Web Application Architecture

Here, we discuss Web applications, not just "Web sites". There is distinction between Web sites and Web applications. By terms, a Web site is a cluster of Web pages, which is composed of a unique node on the Web. Greg Barish [GRE02] pointed that engineers by

habit, tend to associate Web sites with the server side. A web application is more than just server side; there is the network and the client. So, based on this, a Web site (server) is not the same thing as a Web application (the client, network, and server). Jim Conallen [JIM] pointed that if no business logic exists on a server, the system should not be termed as a Web application. The business logic states are affected by user input. Some people consider that a Web site may contain multiple Web applications. For example, yahoo.com web site is composed of multiple applications such as an email application, a calendar application, etc.

In this report, we will loosely define a Web application to be a Web system with Web server, network, HTTP, and browser. We choose to define Web application as any Web-based site or application available on an internet or on an intranet, whether it be a static promotion site or a highly interactive site for banking services.

As showed in Figure 2.1, there are three basic components in the Web architecture: a Web server, a network connection, and one or more client browsers. A Web browser is used as the user interface to the application. It is an application that runs on a client machine. Internet protocols such as HTTP are used for communication between the interface and the application. We call the rest of the application *objects.* The objects reside in the Web server, and user navigates links between Web pages to access these objects. The client sends requests through network connection. Server distributes formatted information to client. The formatted information is stored in files that are sent back to client side. We call the above formatted information *pages* [JIM].

Page is the most fundamental component of a Web application. Web pages make up the user interface for the application [GRE02]. In Web development environment like Microsoft's Active Server Page or Allaire's Cold Fusion, the pages can be a static HTML formatted pages, or dynamic scripted pages, or combined two of them. The scripted pages contain code that is executed by the Web server that accesses server resources to ultimately build an HTML formatted page. The newly formatted page is sent back to the browser that requested it.

Server Scripting

It is important to note that the connection between the client and the server only exists during a page request. Once the request is fulfilled the connection is broken. All activity on the server occurs during the page request. Business logic on the server is only activated by the execution of scripts inside the pages requested by the browser. The server processing includes: update the business state of the server and prepare an HTML formatted page for the requesting browser.

Client Scripting

The browser can execute scripted code in a page. It does not direct access server resources. Typically scripts running on the client augment the user interface as opposed to defining and implementing core business logic. With the acceptance of the new Dynamic HTML specification, client scripts can access and control nearly every aspect of the page's content. When an HTML page is rendered in a browser it is first parsed and

divided into elements. When dynamic HTML is employed each of the elements can be named or assigned an ID, which could be referenced by client side scripts.

Web applications use HTML *form* to collect the user input data. An HTML form is a collection of input fields that are rendered in a Web page. The basic input elements are textbox, text area, checkbox, radio button group, and selection list. *Frames* make it possible to have multiple Web pages active and open at the same time.

Figure 2.2 shows a summary of the major architectural components discussed above.

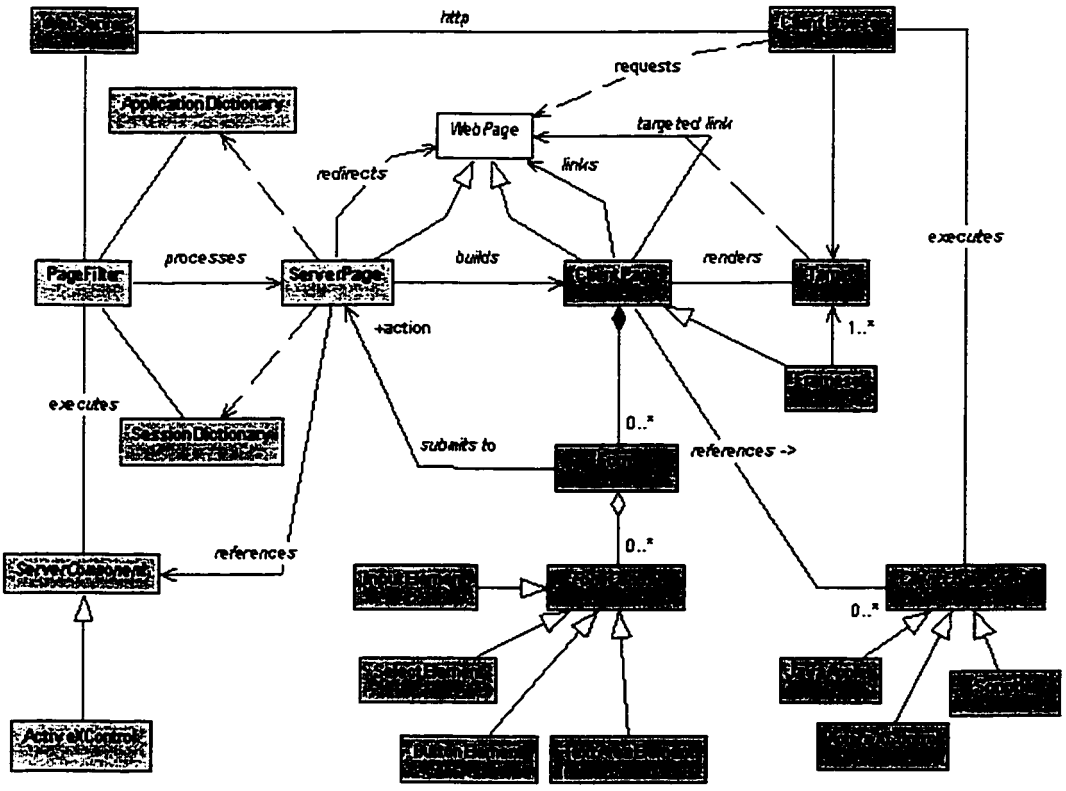

Figure 2.2 Model of a generalized Web application architecture

The **difference between Web infrastructure and traditional client/server architecture** is that:

the processing of a page request in server may result in a process or two to get executed in server, and remain executing long after the client browser shuts down.

The connection between the client and the server is closed once the requested page has been received by the browser.

Understanding the architecture of Web applications is to help the performance evaluation of a Web application. The Web application is not only for server-side solutions it is also concerning with client-side and networking topics because they have a fundamental impact on how end users perceive Web applications. After all, most people who use the Web concern with its end-to-end behavior. For example, if it takes a while to buy a flight tickets online, it maybe caused by a slow modem, an overtaxed server, or network congestion. Measures are not only on that such application might be slow for one user, but also that the system becomes slow as more users access it.

2.3 The Classification of Web Applications

When publishing a Web site, the design is based upon what we hope to achieve with the site. Many researchers have advanced various views of the Internet, and have tried to categorize Web sites from a variety of perspectives. Two major perspectives on Web classification are based on ***technological view*** and ***marketing view***.

Technological perspective

This classification is based on the degree of interactivity the Web sites offers. Thomas A. Powell [POW97] classifies Web sites into five categories:

- *Static Web Sites* - The most basic Web site. Presentation of HTML documents. The only interactivity is to choose page by clicking links.
- *Static with Form-based Web sites* - Forms are used to collect information from the user including comments or requests for information. Main purpose is document delivery, limited emphasis on data collection mechanisms
- *Web sites with Dynamic data access* - Web sites are used as front-end for accessing a database. Via the web page users can search a catalogue or perform queries on the content of a database.
- *Dynamically Generated Sites* - Providing customized pages for every user creates a need to take a step away from the static Web site. The page may be static, providing no interactivity, but the way it is created has similarities with the way screens are created in software applications.
- *Web-based software applications* - Web sites that are part of the business process. This could be an inventory tracking system or a sales force automation tool. The processing of a page request in server may result in a process or two to get executed in server, and remain executing long after the client browser shuts down.

The above classification is derived from the need for methodology of developing a Web site. It is also applicable to the classification of testing process, and performance measurement.

Business prospective

Another classification is based on different business purposes of commercial Web sites. It is focusing mainly on the functions the Web sites can provide for business and marketing. James Ho [HO97] claimed that since technical issues regarding bandwidth and security can be resolved eventually along with the technological development, the more important issue is to ask what (perceived through the customer's perspective) can be created on the Web. Based on his evaluation of 1000 commercial Web sites, he classified commercial Web sites into three categories:

- *Promotion of products and services* - Promotion is the information about products and services that part of the company's business.
- *Provision of data and information* - Provision is the information about, for instance, the environmental care program that the company may sponsor.
- *Processing of business transactions* - Refers to regular business transactions.

Four types of value creation processes have been also identified: *timely*, *custom*, *logistics*, and *sensational*. Ho [HO97] defined timely value as the value of time sensitive information; custom value refers to the customization and personalization of Web sites according to the preferences of the Web visitors; logistic value is defined as predicated on preprogrammed propositions on the Web sites.

This classification is meant to show the purposes with one commercial Web site, it can also be used to categorize the main purpose of a Web site. For example, a company's on-

line catalogue would be a promotional site, a private person's homepage may be a provisional site, and banking services Web site may be considered as a site for processing.

Technical classification tells us nothing about the content of the Web sites. Classifying Web sites based on their economic activities together with the technical view provide information about interactivity and purpose, which gives us an idea on the Web site's complexity.

2.4 Challenges for Web Application System

As we saw in above sections, Web application is a really new type of application, which is based on Internet infrastructure. Internet technologies as a medium for application system make challenges on the performance of the system [GRE02]. Firstly, the users for Web applications are from all over the world wider than any other medium. The Web is active: users not only receive information, they also submit it. Secondly, the Web is a dynamic medium where a user request does not always get the same server-side response. Although the Internet utility is named "always on", but there are no guarantees about when and how often user's information can be accessed. Thirdly, there is no complete control on the information delivered to end-users. The above characteristics of the Internet make challenges in the Web application development under Internet environment, especially with the introduction of electronic business. Here are some:

- Increasing importance

Internet infrastructure requests that Web application systems are not merely to automate the well-established tasks and processes but also to redesign business processes, and even to create entirely new businesses and industries.

- Growing complexity

  The complexity of the Web application system has increased and is still increasing [DIR02]. The tools and technologies for the development process are getting larger and in many cases they are not compatible with each other. For example, the traditional programming languages like COBOL, C and C++ are continued to be used beside newer languages like Java, VB.NET and C#. The multitude of available platforms gives more design choices in systems architecture. This increases the importance of the integration of heterogeneous systems and the management of interfaces associated with them. The complexity of Web systems is increasing due to new and additional application areas where more complex problems are being addressed.

- Increasing quality requirements

  With the complexity increasing, the quality requirements of Web application systems have increased importantly. Because Web systems are applied in more important and critical areas than before, the failure of such systems can result in serious economical and material damages. Especially quality features like security (access control, authentication, encryption) and reliability (availability, fail-safeness, correctness) must be essential parts of a Web service environment today.

Chapter 3

Introduction to Software Measurement

Measurement in science has a long history. Today, computers play a primary role in almost every area of our life. It increases the important requirements on software measurement. All successful software organizations implement measurement as part of their day-to-day management and technical activities. Measurement provides the objective information they need to make informed decisions that positively impact their business and engineering performance. Software measurement has evolved into a key software engineering discipline. In the past, measurement was treated as an additional, non-value-added task. Now measurement is considered to be a basic software engineering practice.

3.1 Basic Concepts of Software Measurement

Fenton [FEN97] describes measurement as following:

"Measurement is a process by which numbers or symbols are assigned to attributes of entities in the real world so as to describe the attributes according to clearly defined rules."

An *entity* is an object or event in the real world. For example: a person, a race. *Attributes* are features or properties of entities. For example: height of a person, time taken to

complete a race. Measurement mapping *M* must map entities into numbers and empirical relations and numerical relations in such a way that the empirical relations preserve and are reserved by the numerical relations.

Software measurement is one of the areas in Software Engineering that is to apply an engineering approach to the construction and support of software products so they can safely fill the uses to which they may be subjected. Software measurement in software engineering provides feedback and assists the software development and maintenance. Software measurement is also known as software metrics or software measures. In software engineering, the term **software metric** is often used.

A **software metric** is a simple quantitative measure derivable from any attribute of the software life cycle. Yet, in the realm of software engineering and mathematics, not everything is quantitative. Large part of mathematics, including most of logic is not quantitative [BER98]. Thus, we would like to present software metrics as following:

Software metrics *use mathematical presentation to quantify software development process, product and resources. It offers an efficient way for software engineers to control software development life cycle---to value the collections of software-related activities; measure artifacts, deliverables or documents and distribute software entities.*

Ordinarily and traditionally, the measurement of the software process and product are studied and developed for use in predicting or evaluating the software development

process. *Productivity*, *product quality* and *product costs* are examples that the metrics applied to. Information gained from these metrics and models can then be used in management and control of the development process, leading one hope to attain overall improvement. From another point of view, software design consumes resources and produces a product. Thus, the measurements of resources are important. Metrics on personnel, software and hardware including performance are used to help the effectiveness of organization.

Good metrics should not just stay on the description level. Instead, the actual prediction or estimation of process, product and resources should be presented. Hence, a good software metrics should have the quality of simple, robust, easy of maintain, valid and objective (to the greatest extent possible). Moreover, metrics should have data values that belong to appropriate measurement scales for maximum utility in analytic studies and statistical analysis [CON86].

In the terminology of measurement theory often use the term measures. Software quality should be linearly related to a software measure. **Software measures** are mappings from objects in software entities to numbers or symbols to quantify a software attribute. A software measure in general should be objective, reliable, valid, and robust. *Objectivity* means that the measurement process should not depend on the subject (person, tool) that performs the measurement, on system's size, or on the programming language. The *reliability* requires the metrics to characterize in a unique way every entity measured. Equal entities should obtain equivalent measurement values. Repeated measurement in

equal conditions should give same values for the same entities. The *robustness* requires the ability of a measure to tolerate incomplete information. The software measure requirement *validity* means that the measurement data should reflect exactly the characteristics of the entity under measurement.

Like any engineering activity, software measurement requires a definition of the environment in which the measurement is expected to perform. To set the environment, we need to know whom we measure for, why, what we measure, when we measure it, and what measures to use within a context of an organization or specific project.

Why measure software

In the area of software engineering, measurement has been discussed for many years. The magnitude of costs involved in software development and maintenance increases the need for a scientific foundation to support programming standards and management decisions by measurement.

What we can measure

The software measurement activities are based on the classification of software entities. There are three classes of software entities whose attributes we may want to measure:

- *Processes* - the set of activities used by an organization to develop its products. The attributes are, for example, cost-effectiveness and effort.
- *Products* - the deliverables created during the course of a project. They include requirements, functional specifications, design documentation, source code, test

cases, test results, etc. Products are measured directly by examining their intervals. Products are measured indirectly by examining their behavior in specific situations.

- *Resources* - the input to the process used on a project (i.e., people, tools, materials, methods, time, money, and products from other projects). Some of the resources attributes are the productivity of the team and capability-maturity assessment of the personnel and tools.

Software entities distinct between attributes are internal and external:

- *Internal attributes* are those attributes that can be measured purely in terms of the process, product, or resource itself.
- *External attributes* are those attributes that can only be measured with respect to how the process, product, or resource relates to its environment.

External product attributes include *quality, reliability, testability, reusability*, and *maintainability*. Internal product attributes include size, complexity, reuse, defects, coupling, cohesion and polymorphism.

How to measure

The essential goal of software measurement is to identify anomalies within the same development phase in which it originated, as well as to measure progress. Thus, each software development life-cycle phase should contain evaluative metrics to achieve high project visibility and quality control. The prediction measurement should be applied from

within the early phases of software development to predict future characteristics of software entities. A set of metrics has to be collected meaningful measurement data and to analyze it according to clearly defined rules.

3.2 Software Metrics classification

Only a goal of measurement determines the appropriateness of software measures. There are many ways to classify software metrics. With respecting to software design life cycle, software metrics can be classified as *product metrics*, *process metrics*, and *resources metrics*. Same to the definition, the general subdivision of software metrics varies. We present one popular classification based on the work of Fenton [FEN97] and Meyer [BER98] in Table 3.1. Product metrics is measures of the software products at any stage of the life cycle, from requirements to system delivered. Process metrics are measures of the process used to obtain these products, such as overall development time, type of methodologies used and the customer satisfaction levels; resources metrics are measures of the behavior and the development environment.

Process Metrics	**Maturity Metrics** – organization ~, resource ~, tech-management ~, document standards ~, data-management and analysis ~		
	Management Metrics 1. *Project management* (milestone ~, risk ~ workflow ~, controlling management database ~) 2. *Quality management* (customer satisfaction ~, review ~, productivity ~, efficiency ~, quality assurance ~) 3. *Configuration Management* (change control ~, version control ~)		
	Life Cycle Metrics – problem definition ~, requirement analysis and specification ~, design ~, implementation ~, maintenance ~		
Product Metrics	**Internal**	*Size metrics*	Providing measures of how big a product is internally
		Complexity metrics	Assessing how complex a product is
		Style metrics	Assessing adherence to writing guidelines for product components
	External	*Product reliability metrics*	Assessing the number of remaining defects
		Functionality metrics	Assessing how much useful functionality the product provides
		Performance metrics	Assessing product's use of available resources, computation speed
		Usability metrics	Assessing a product's ease of learning and ease of use
		Cost metrics	Assessing the cost of purchasing and using a product
Resources Metrics	**Personnel Metrics** – Programming experience ~, Communication level ~, Productivity ~, Team Structure ~		
	Software Metrics – Performance ~, Paradigm ~, Replacement ~		
	Hardware Metrics – Performance ~, Reliability ~, Availability ~		

Note: here, we use ~ to denote metrics

Table 3.1 Classification of Software Metrics

Another way to classify software metrics is by the computational methodologies used. Grady [GRA87] pointed out this as primitive metrics and computed metrics. Primitive metrics are those that can be directly observed, such as the program size (in LOC), number of defects observed in unit testing, or total development time for the project. Computed metrics are those that cannot be directly observed but are computed in some manner from other metrics. Computed metrics is normally used for productivity. Such as

LOC produced per person-month (LOC/person-month). Grady indicated that computed metrics is combinations of other metrics and thus are often more valuable in understanding or evaluating the software process than that of primitive metrics.

3.3 Software Metrics

In this section, we introduce the common used software metrics today. Since object-orient approach is becoming more and more popular in today's software development, there is a need to distinguish O-O design from traditional procedural programs. Object oriented system requires not only a different approach to design and implementation, but also a different approach to software metrics. Therefore, we classify software metrics as Object-Oriented metrics and traditional metrics. Traditional metrics is applied in procedure-oriented programs, which typically include Cyclomatic Complexity (CC), Lines of Code (LOC), Comment percentage (CP), Halstead's software metrics and etc. Whereas, object-oriented metrics mainly apply to evaluate object-oriented design in the following key features such as classes, objects, methods, coupling, cohesion, inheritance, polymorphism, encapsulation, and etc. From the point of effectiveness, researches ([TEG92], [LIN98]) have shown that a combination of selected traditional and object-oriented metrics provides the best results when we analyzing the overall quality of object-oriented system.

Software Metrics techniques were introduced in software area 30 years ago to measure the complexity of traditional program written by FORTURN, COBOL, C, etc. Traditional software complexity metrics are measures of the ease or difficulty of a programmer

performing common programming tasks such as testing, understanding, or maintaining a program. Complexity metrics do not measure the complexity itself, but instead measure the degree to which those characteristics lead to complexity exist within the code and the degree to which those code characteristics occur in the code impact the ease or difficulty of a programmer working with the code. Some traditional complexity metrics still used widely and can be applied to object-oriented programs, such as Line Of Code (LOC), McCabe's Cyclomatic Complexity.

3.3.1 Traditional Software Metrics

Lines of Code (LOC) [LIN98]

LOC is a widely used metric for program size. Size of a program is used to evaluate the ease of understanding by developers and maintainers. Size can be measured in a variety of ways, which include counting all physical lines of code, the number of statements, the number of blank lines, and the number of comment lines. Lines of Code (LOC) count all lines. Non-comment Non-blank (NCNB) is sometimes referred to as Source Lines Of Code and counts all lines that are not comments and not blanks. Executable Statements (EXEC) is a count of executable statements regardless of number of physical lines of code. For example, in C, IF statement may be written as:

```
if (i < 20 && i >0){
   X = 10;
}
```

Here, there are 3 LOC, 3 NCNB, and 1 EXEC.

Cyclomatic Complexity - v(G)[LIN98][ALE98]

McCabe Cyclomatic complexity is to compute a number v(G) where G stands for the associated graph of the flowchart. v(G) refers to the number of edges minus the number of the nodes in the flowchart. Here, in flowchart graph, nodes mean the statements and decision boxes; edges mean the links between them. The cyclomatic complexity of such a graph can be computed by a simple formula as following as:

$$v(G)=e-n+2,$$

e: the number of edges; *n: the number of nodes in the graph;*

Take, for example, the flowchart in figure 3.1. Here, e=8, n=7, then v(G)=e-n+2=8-7+2=3.

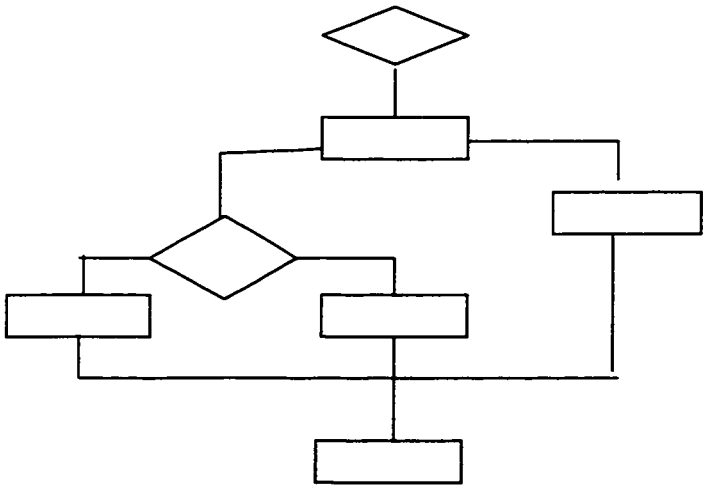

Figure 3.1 Cyclomatic complexity flowchart with edges and nodes

Cyclomatic complexity can be used as a measure of program complexity and as a guide to program development and testing [MCC94]. An algorithm with a low cyclomatic complexity is generally better, but this may imply decreased testing and increased understandability. Thus low cyclomatic complexity is not good for program comprehension and maintenance.

Halstead's software metric

Halstead's software metric is used to estimate the number of errors in the program. It uses the number of distinct operators and the number of distinct operands in a program to develop expressions for the overall program length, volume and the number of remaining defects in a program. The length of the program is estimated:

$$N = N1 + N2$$

Where *N1 = total number of operators occurring in a program*
N2 = total number of operands occurring in a program
N = length of the program

And the Volume of the program is estimated:

$$V = N\log_2(n1 + n2)$$

Where *V = volume of the program*
n1 = number of unique or distinct operators in a program
n2 = number of unique or distinct operands in a program

Thus, we have:

$$N1 = n1\log_2 n1$$
$$N2 = n2\log_2 n2$$

There are tow empirical formulae proposed by Halstead to estimate the number of remaining errors in a program, respectively, are:

$$\dot{E} = \frac{V}{3{,}000} \quad \text{and} \quad \dot{E} = \frac{A}{3.000}$$

where

$$A = \left(\frac{V}{\frac{2n_2}{n_1 N_2}} \right)^{\frac{2}{3}}$$

E = number of errors in the program

Halstead's theory of software metric is probably the most well known technique to measure the complexity in a software program and the amount of difficulty involved in testing a debugging the software.

3.3.2 Object-Oriented Software Metrics

To make use of the measurement during the OO design process, the components of the design have to be measured directly in order to assess the course of the design.

Measurable Internal Attributes of OO System:

- *Coupling in OO System*
  - *Between Classes*
  - *Within a Class*
- *Cohesion in OO System*
- *Inheritance in OO System*
- *Polymorphism in OO System*
- *Size in OO System*

Object Oriented good development style rules:

- *To minimize* the coupling between classes
- *To increase* cohesion
- *To increase* polymorphism

Coupling metrics

In the context of object-oriented paradigm, coupling describes the interdependency between methods and between object classes, respectively [JOH92].

CBO (Coupling between object classes) [SHY94][SHY91]

CBO for a class is a count of the number of other classes to which it is coupled. An object is coupled to another object if one of them acts on the other, i.e., methods of one class use methods or attributes of another, or vice versa. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

Coupling Factor (COF) [FER96][RHA98][FBA96]

Coupling factor (COF) is defined as the ratio of the actual number of couplings to the maximum possibly number of couplings in the system, excluding coupling due to inheritance. COF checks whether two classes related either by message passing or by semantic association links (reference by one class to an attribute or method of another class).

$$COF = \sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client\ (C_i, C_j)\right] / (TC^2 - TC)$$

Where:

$$\text{is\_client}(Cc, Cs) = \begin{cases} 1 & \text{iff } C_c \Rightarrow C_s \wedge C_c \quad C_s \\ 0 & \text{otherwise} \end{cases}$$

Cc => Cs represents the relationship between a client class Cc and a sever class Cs

TC is the number of classes

TC2-TC refers to the maximum number of couplings in a system

Coupling metrics from Briand et al [LIO97]

These measures focus on coupling as caused by interaction that occurs between classes [LIO96]. The measures are built on three facets by relationship between classes (friendship, inheritance, and other), different types of interactions (attribute-class, class-method, and method-method), and the locus of impact of the interaction (import, export). Therefore, based on the combination of the above three facets, there are 18 different metrics such as IFCAIC, ACAIC, OCAIC, FCAEC, DCAEC, OCAEC, IFCMIC, ACMIC, OCMIC, FCMEC, DCMEC, OCMEC, IFMMIC, AMMIC, OMMIC, FMMEC, DMMEC, OMMEC. Definitions of this metrics are presented in the following table.

Briand et al coupling metrics [LIO97]	
Name	**Definition**
IFCAIC: Inverse friend CA import coupling	These metrics count for the interactions between classes. The first or first two letters represent the type of relationship considered (i.e., IF for Inverse Friend, F for Friend, D for descendant, A for ancestor, O for others). The 2 letters afterwards capture the type of interaction (i.e., CA for class-attribute interaction, CM for class-method interaction, MM for method-method interaction). The last 2 letters say the locus of impact. IC counts for import coupling, and EC counts for export coupling. (i.e., Import: class c is the client class in the interaction; Export: class c is the server class in the interaction.)
ACAIC: Ancestors CA import coupling	
OCAIC: Others CA Import coupling	
FCAEC: Friends CA export coupling	
DCAEC: Descendant CA export coupling	
OCAEC: Others CA export coupling	
IFCMIC: Inverse friend CM import coupling	
ACMIC: Ancestors CM import coupling	
OCMIC: Others CM import coupling	
FCMEC: Friends CM Export coupling	
DCMEC: descendant CM export coupling	
OCMEC: Others CM export coupling	
OMMIC: Others MM import coupling	
IFMMIC: Inverse Friend MM Import coupling	
AMMIC: Ancestors MM import coupling	
OMMEC: Others MM import coupling	
FMMEC: Friends MM export coupling	
DMMEC: Descendant MM export coupling	
OMMEC: Others MM export coupling	

Table 3.2 Briand et al coupling metrics

- The higher the export coupling of a class, the greater the impact of a change to the class on other classes.
- The higher the import coupling of a class C, the greater the impact of a change in other classes on C itself.

- Coupling based on friendship between classes is in general likely to increase the likelihood of a fault even more than other types of coupling, since friendship violates modularity in OO design.

Cohesion metrics

Cohesion describes the binding of the elements within one method and within one object class, respectively [JOH92]. Classes with strong cohesion are easier to maintain, and furthermore, they greatly improve the possibility for reuse.

LCOM (lack of cohesion in methods) [SHY94]

The LCOM is a count of the number of method pairs whose similarity is zero minus the count of method pairs whose similarity is not zero. Here, the degree of similarity based on the attributes in common by the methods. The larger the number of similar methods, the more cohesive the class is.

- Cohesiveness of methods within a class is desirable, since it promotes encapsulation.
- Lack of cohesion implies classes should probably be split into two or more sub-classes.
- Any measure of disparateness of methods helps identify flaws in the design of classes.
- Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

Bieman and Kang's class cohesion measures [LIM98][FER96]

Bieman and Kang concern the cohesion among class components including attributes and classes. Their class cohesion measures are based on the direct or indirect connectivity between a pair of methods. Two methods are directly connected if they use one or more common attributes. In contrast, two methods that are connected through other directly connected methods are indirectly connected.

Here, NDC(C) is the number of directly connected methods in a class C. NIC(C) is the number of indirectly connected methods in a class C. NP(C)=N* (N-1)/2 is the maximum possible number of connections in a class.

Tight Class Cohesion (TCC) is defined to be a ratio of the number of directly connected methods in a class, NDC(C), to the maximum possible number of connections in a class NP(C).

$$TCC(C) = NDC(C)/NP(C)$$

Loose Class Cohesion (LCC) is defined to be a ratio of all directly connected methods, NDC(C), and indirectly connected methods, NIC(C), in a class to the maximum possible number of connections in a class, NP(C).

$$LCC(C)=(NDC(C)+NIC(C))/NP(C)$$

Inheritance metric

Inheritance is a reuse mechanism that allows programmers to define objects incrementally by reusing previously defined objects as the basis for new objects [VIC01].

Depth of inheritance tree (DIT) [SHY94]

The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor class [LIN98].

- The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior.
- Deeper trees constitute greater design complexity, since more methods and classes are involved.
- The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.

Number of children (NOC) [SHY94]

The number of children is the number of immediate subclasses subordinated to a class in the class hierarchy. It is an indicator of the potential influence a class can have on the design and system [LIN98].

- Greater the number of children, greater the reuse, since inheritance is a form of reuse.
- Greater the number of children, the greater the likelihood of improper abstraction of the parent class is. If a class has a large number of children, it may be a case of misuse of sub-classing.
- The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

Method Inheritance Factor (MIF) [FER96][RHA98][FBA96]

The Method Inheritance Factor is defined as the ratio of the number of inherited methods to the total number of the methods that can be invoked in association with the classes in a system. Here, for each class C1, C2...Cn, a method counts as 0 if it has not been inherited and 1 if it has been inherited.

$$MIF = \sum_{i=1}^{TC} M_i(C_i) / \sum_{i=1}^{TC} M_a(C_i)$$

Where

Mi(Ci) is the number of methods inherited(and not overridden) in Ci

Ma(Ci) is the number of methods that can be invoked in association with Ci

TC is the total number of classes

Attribute Inheritance Factor (AIF) [FER96][RHA98][FBA96]

AIF is defined as the ratio of the number of inherited attributes to the total number of available attributes (locally defined plus inherited) for all classes in a system.

$$AIF = \sum_{i=1}^{TC} A_i(C_i) / \sum_{i=1}^{TC} A_a(C_i)$$

Where

Ai(Ci) is the number of inherited attributes in Ci

Aa(Ci) is the number of attributes that can be invoked associated with Ci

TC is the number of classes

The larger of the metrics values, the greater the number of methods or attributes it is likely to inherit, making it more complex to predict its behavior.

Polymorphism metrics

Polymorphism means having the ability to take several forms. For object-oriented systems, polymorphism allows the implementation of a given operation to be dependent on the object that contains the operation [VIC01].

Polymorphism Factor (POF) [FER96][RHA98][FBA96]

POF is the number of methods that redefine inherited methods, divided by the maximum number of possible distinct polymorphic situations (the latter represents the case in which all new methods in a class are overridden in all its derived classes). Thus, POF is an indirect measure of the relative amount of dynamic binding in a system.

$$POF = \sum_{i=1}^{TC} M_o(C_i) \Big/ \sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]$$

Where

Mn(Ci) is the number of new methods

Mo(Ci) is the number of overriding methods

DC(Ci) is the descendants count(the number of classes descending from Ci)

TC is the number of classes

Polymorphism metrics by Benlarbi and Melo [SDI99]

Overloading

Overloading means methods have same names with different signatures within a class scope. The OVO metrics is to gauge the degree of methods generality in a class by counting the number of member functions that implemented the same operation such as

add "+ ", minus "-" operation. Here, overl(fi, C) counts the number of items that the function member fi is overloaded in the class C.

$$OVO = f_i \circ_C overl(f_i, C)$$

Static polymorphism

Static polymorphism refers to method overriding. That is, methods have the same name and different signatures among inheritance related classes, and the corresponding binding occurs in compile-time. Spoly(Ci, C) counts the number of static polymorphism functions that appear in Ci and C.

Static polymorphism in ancestors:

$$SPA(C) = C_i \circ_{Ancestors(C)} Spoly(C_i, C)$$

Static polymorphism in descendents:

$$SPD(C) = C_i \circ_{Descendents(C)} Spoly(C_i, C)$$

Dynamic polymorphism

Dynamic polymorphism occurs in the same name and same signature in an overridden method, and the corresponding binding occurs in run-time. Dploy(Ci, C) counts the number of dynamically polymorphic functions that appear in Ci and C.

Dynamic polymorphism in ancestors:

$$DPA(C) = C_i \circ_{Ancestors(C)} Dpoly(C_i, C)$$

Dynamaic polymorphism in descendents:

$$DPD(C) = C_i \circ_{Descendents(C)} Dpoly(C_i, C)$$

Encapsulation metrics

Encapsulation means separating the external aspects of an object that are accessible to other objects, from the internal implementation details of the object that are hidden for other objects. Encapsulation prevents a program from becoming interdependent that a small change has massive effects. [VIC01]

Method Hiding Factor (MHF) [RHA98][FBA96]

MHF is defined as the ratio of the number of the invisibility of all methods to the total number of methods declared in all classes. The invisibility of a method is the percentage of the total classes from which this method is not visible.

$$MHF = \sum_{i=1}^{TC} \sum_{m=1}^{Md(Ci)} (1-V(M_{mi})) / \sum_{i=1}^{TC} M_d(C_i)$$

Where:

$$V(M_{mi}) = \sum_{j=1}^{TC} is\_visible(M_{mi}, C_j) / (TC-1)$$

$$is\_visible(Mmi, Cj) = \begin{cases} 1 & \text{iff } j \neq i \wedge C_j \text{ may call } M_{mi} \\ 0 & \text{otherwise} \end{cases}$$

Md (Ci) is the number of methods declared in a class (not inherited)

TC is the number of classes

Attribute Hiding Factor (AHF) [RHA98][FBA96]

AHF is defined as the ratio of the number of the invisibility of all attributes to the total number of attributes defined in all classes. AHF was defined in an analogous fashion, but using attributes rather than methods.

$$AHF = \sum_{i=1}^{TC} \sum_{m=1}^{Ad(Ci)} (1-V(A_{mi})) / \sum_{i=1}^{TC} A_d(C_i)$$

Where:

$$V(A_{mi}) = \sum_{j=1}^{TC} is\_visible(A_{mi}, C_j) / (TC-1)$$

$$is\_visible(Ami, Cj) = \begin{cases} \text{iff } j \quad i \wedge C_j \text{ can reference } A_{mi} \\ 0 \quad \text{otherwise} \end{cases}$$

Ad (Ci) is the number of attributes declared in a class (not inherited)

TC is the number of methods declared in a class

Other Metrics

Weighted methods per class (WMC) [SHY94][SHY91]

WMC is a count of the complexities of the methods implemented within a class (Complexities here refers to Cyclomatic Complexity). WMC belongs to the popular CK metrics suite and reflect the complexity of classes.

WMC Considers a Class C1, with methods *M1, ... Mn* those are defined in the class. Let c1, ..., cn be the Cyclomatic Complexity of the methods. Thus, $WMC = \sum_{i=1}^{n} C_i$.

The number of methods and the complexity of methods involved is a predictor of how much time and effort is required to develop and maintain the class.

- The larger the number of methods in a class the greater the potential impact on children, since children will inherit all the methods defined in the class.
- Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse.

So far, the metrics developed by Chidamber and Kemerer [SHY94][SHY91] are the most popular ones, because they are widely referenced, and most commercial metrics collection tools available include the metrics.

Even though software metrics has been a subject area over 30 years, it has barely penetrated into mainstream software engineering. Having exploring research of software metrics, we observe two intuition reasons. One is that most of metrics has been defined by an individual or a team and then tested and used only in a very limited environment, thus it is hard to achieve the adoption in software industry. Another is that most of software metrics has not addressed their most important requirement: to provide information to support quantitative managerial decision-making during the software lifecycle [NOE00].

3.4 Summary

In this chapter, we survey software measurement by reviewing the key concepts in software measurement, and introducing the selected popular traditional and object-oriented metrics.

Chapter 4

Quality Assurance in Web Application Project

4.1 Introduction

As Web is playing a central role in diverse application domains, there are strong reasons for a systematic and disciplined use of engineering methods and tools for developing and evaluating Web sites and Web applications [MUR01]. Nowadays is widely accepted that a Web site is not merely a matter of contents, it also requires demanding functionality, combining navigation with complex services and transactions. Because of the increasing size, complexity, quality needs and market needs for Web applications, several problems have frequently been reported, such as: exceeding budgets, systems didn't meet business requirements, unknown or bad product quality, and lack of requirements and architectural documentation [CUT00]. Besides, the quality of Web applications has often been assessed in an ad-hoc way, mainly based on the common sense and experience of developers. Evaluation methods and techniques have been developed for different assessment purposes [FLE00][MEN01][POO02], such as Web Quality Evaluation Method (QEM) [OLS00] and its supporting tool, Web QEM tool [OLS01]. They focus on those aspects that are perceived by the user, like navigation, interface, reliability, etc. instead of other product attributes, such as quality of code, design, etc.

However, developing products and services meeting the customers' needs is always the main target. Today, the end-client not only asks for a working product, but also for the

development of high quality products. The technology progress and the worldwide competitor make the enormous increase of application complexity. It also leads to develop an application in even shorter period of time to meet the market needs. In this situation, to develop successful products means to be successful on the market. Quality is one of the most important factors for the commercial success of a software system (especially when looking at its whole life span, including maintenance activities). Therefore, integrating the future customers of the application and quality control in very early phase of development becomes a must. Based on the above, user-oriented design and quality are introduced to the development processes, where end user plays an important role. For example, at Berkom, the application-oriented innovation center of T-Systems Nova in Germany, the approach of user-centered quality engineering has been introduced in Web application development [MIC02]. In this approach the user and their requirements are integrated in the development processes. By applying this approach within development phases, it gained that efficiency is increased, costs reduced and the end-user acceptance and satisfaction with the application increases. The reason is obvious since the needs of the customer and the market are met.

Customers' needs and satisfaction are always the most important factors in the quality of today's software application, especially in Web projects. Customers want what they want, when and how they want it. If they don't get instant satisfaction they may turn around to other company to spend their money and time. Poor service and quality can be a kind of risk in Web project since the effect is that we cannot keep the customer. Web development in general has to identify and assess these risks. QA and testing can be used

to identify the risks and ensure to minimize the risks in a Web project. This should be part of the Web application development throughout its life cycle.

In this chapter, we give an overview of QA in the Web project development processes. **Section 4.2** gives a brief overview of Web project development processes. **Section 4.3** introduces an approach in applying user-centered quality engineering in Web application design processes, which developed at Berkom. **Section 4.4** introduces how to establish quality procedures in software development based on an accounting project developed in Dresdner Bank AG (Germany) which using the framework based on e-commerce strategy. **Section 4.5** describes how to use QA and testing to control risks in various phases of Web development.

4.2 Web Application Development Processes

When we evaluate a Web application, the evaluation needs to locate various development phases. Web Application system is different from the traditional system as we saw in Chapter 2. In e-commerce, the system providing the Web-based service includes business processes, network communications, various software components and database engines. Thus we can address four topics for a Web-based system: *Customer's Needs*, *Business Processes*, *Web-based System Architecture*, and *Web Software Components*. Thomas Fehlmann [AND02] gives a Web development model based on the well-known software development *V-model.* Figure 4.1 shows this model. The significant difference is in this Web development model, testing is the first task on each of the development phases and the last task as well. This model gives on various topic levels and can be extended to an

additional upper topic level that represents the needs of the Web users. Based on this model, the development processes can split into different level, which corresponds to one of the above four topics.

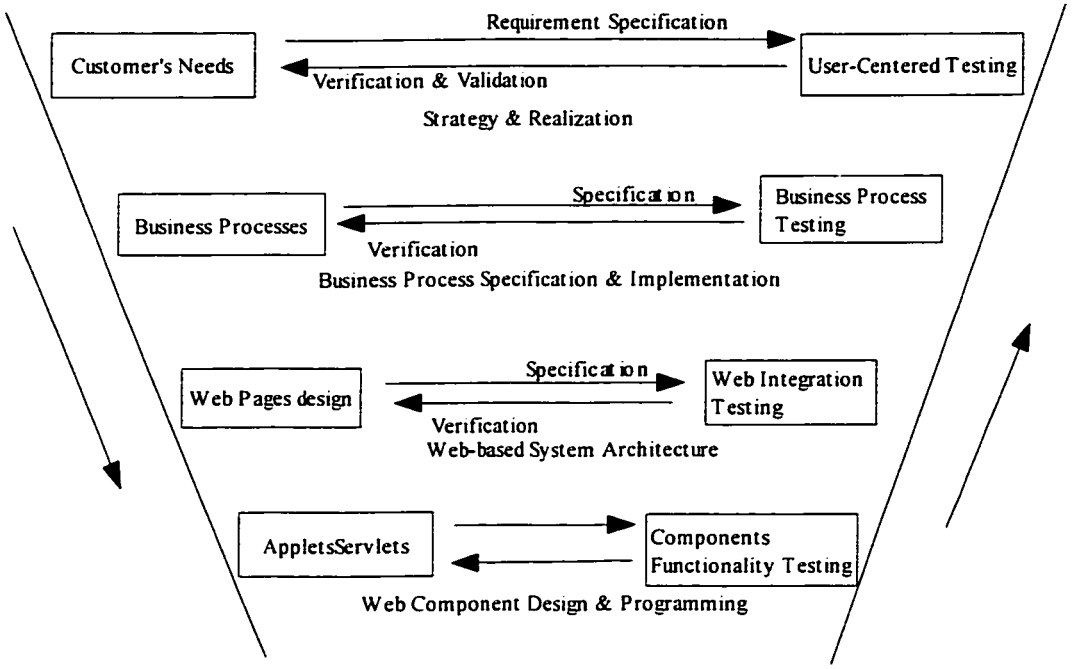

Figure 4.1: Web Development Model

The top level is the first to start from the initial requirements for the Web systems. Web design and integration define how the Web application system will offer its services.

Within the scope of the development process, there are different stakeholders of each process whom have to be considered. The roles in one development phase maybe customers, project manager, etc, but, on the other hand, in other process cycles, they maybe developer, tester, and so on. In a user-centered development environment, the user-related roles are distinguished. Following lists the customer-related roles, which are

described because they are especially important in the scope of user-centered quality engineering [MIC02] that we will see in next section:

- *Customer*: this is for example the project manager who wants to create a new application or a service.
- *End-user*: a member of the target group for which the application has been developed and who wants the service as a buyer on the market. This role is very important in a B2B (business-to-business) environment.
- *User*: the person who finally uses the application or service. This can be for example the buyer's child using learning software or the customer of the service operator. In some cases, like a standard consumer business, the customer and the user would be presented in one person.

Figure 4.2 shows an example of the relation of these roles in B2B environment.

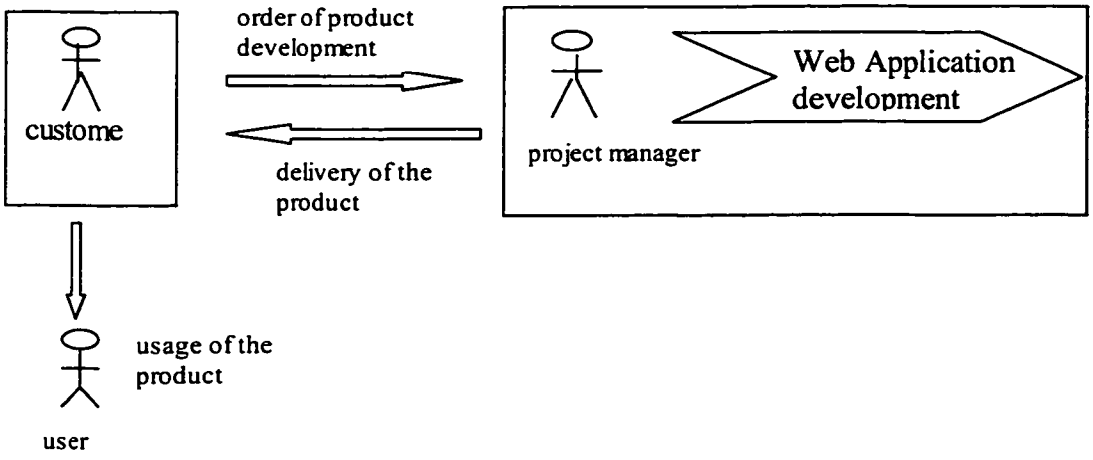

Figure 4.2: Customer roles in a B2B scenario

4.3 User-Centered Quality Engineering in Processes [MIC02]

In a fast-developing market, it is absolutely necessary that the software product fulfill the requirements of users as well as good level of quality. The logical consequence is to

integrate the user requirements and also the potential users in the development processes. In Berkom, engineers introduced user-centered quality engineering into their development process structure. By applying this approach, it solved the quality questions relating to acceptability, usability, and quality of service. As a result, the end-user acceptance and satisfaction, and product efficiency have increased.

4.3.1 Basic Concepts [MIC02]

The goals of user-centered quality engineering are:

- To know what users want.
- To know what users need.
- To avoid expensive redesigns

In summary, the goal of user-centered quality engineering is to design user-friendly and well-accepted applications and services.

Usability, acceptability and *quality of service* are the three quality factors considered in user-centered quality engineering, which are user relevant factors.

Usability

Usability concentrates on the use of products and services in a user point of view. Usability engineering allows optimize the design of applications and services. Style guides and CI/CD regulations as well as usability guidelines (e.g. from ISO) for the design of software and hardware provides the necessary guidelines.

Acceptance

Acceptability covers the users' sense of the whole product and their willingness to use it. A set of factors can influence the acceptance. In user-centered quality engineering, there are classifies as product-specific, user-specific and environment-specific factors. In B2B market, there is also enterprise-specific factor such as organization and qualification of staff.

Product-specific factors are those which can be influenced before, during or after product development. They include functions of the product, its usability, design and additional features, the cost and the price. Product-specific factors are determined by user-specific and environment-specific factors. User-specific factors are, for example, the user's readiness to take risks, the added value and the affinity for technology. The environment-specific factors include innovation cycles and advertising. Table 4.1 gives the technical innovations of acceptability.

Product-specific	Usability, technological complexity, compatibility Charges
User-specific	Information standard, risk readiness, predilection to technology
Environment-specific	Accelerated cycles of innovations
Enterprise-specific	Organization, manner of implementation, qualification of staff, intensity of competition

Table 4.1: Acceptability of technical innovations

Quality-of-Service

Quality-of-Service (QoS) analysis is the evaluation of the user-perceived transmission quality of speech, audio, video, and audio-visual through user tests of instrumental

methods. Here is as example the question of the end-to-end quality of an IP telephony connection as perceived by the user. QoS parameters in the IP network can be an indicator for the quality, but how much they leave open or in how far which influence on quality have to be considered also.

Integrate User-centered Quality Engineering into development processes

To integrate the above three quality factors in the development process of an application, from the idea to the finished product, is the core of user-centered quality engineering that supports product and marketing management and development processes. The development cycle here is divided into four stages:

- **Product Idea (Strategy & Realization)**

  In this stage, new application requirements are discussed with potential end-users under different viewpoints. They include questions related to the acceptance and further design of the product, possible functions and features, quality requirements and user needs or possible target groups. Thus, important information regarding to further specifications and possibly a first feeling for the market can be obtained.

- **Product development (Specification & Implementation)**

  The requirements of the future users or the market obtained from the first stage can be included in the development process. Support can be given by means of usability reviews, acceptance tests or quality of service examinations of partial implementations, pre-products or prototypes.

- **Product Initiation**

  Before the product is introduced on the market, it is necessary to test the product regarding its suitability for the market. Apart from evaluating the product and its features, the future potential improvement can be identified in this stage. Examinations at this time can also be used to restrict the target group for the product or the service.

- **Product on Market**

  Once the product has been put on the market, the users' feeling and satisfaction with the product and the features are especially important. The main goal here is the analysis of the product on the market and to find feedback for future improvement.

4.3.2 Integration of User-Centered Quality Engineering and Quality Criteria for Processes

In Berkom, the approach of user-centered quality engineering has been developed and integrated in the development processes of several projects. This is a three-stage approach:

1. *Analysis*
2. *Integration*
3. *Review*

Analysis

This first stage is to analysis the current process structures and their use. The goal is to identify approaches for integrating the measures regarding user-centered quality

engineering. Then the elements of user-centered quality engineering can be integrated into the development process with their description (roles, documents, etc.).

To start the assessment, it needs to set the topics. This is based on different projects. For instance, the Berkom projects focussed on following topics:

- Web-based projects.
- The support of customer, end-customer and user, e.g. user surveys dealing with usability, acceptability and quality-of-service surveys.
- Different sizes and duration
- Initiated by the main customers

It may need to choice an assessment tool to obtain the representative results. According to above focuses, the following processes can be chosen from the Process Structure:

- *Project management* (a general topic to understand the project and to find out how the project is managed).
- *Quality management* (to learn how the quality of the products and project processes is managed).
- *Quality assurance* (to discover how the project assures the quality of their products and the QA methods used).
- *Customer Need Management* (to learn how the customer and user demands and expectations are understood and managed with special focus on integration of the customer/user into the lifecycle).
- *Process design* (to find out how projects deal with their processes)

Analysis of the assessment reports, designers in Berkom could derived the relevant quality criteria for the introduction of user-centered quality engineering into their project process structure:

- *Flexibility* and *customizability:* Since the process had to fit various projects of very different sizes, the additional features had to be introduced in a way that the processes are easy to use, easy to customize and therefore applicable to all relevant project types and sizes.
- *Supporting* project managers and development teams.
- *Completeness* with respect to different QA techniques and internal services.
- *Open design* and *good supportability.*
- *Reduction* of development costs and *shorten* the development time.
- *Increase* of the product complexity and quality of products and services.
- *Increase* of the customer satisfaction.

Integration

The integration means to integrate user-centered quality engineering into the development model the project uses, like the Web development model in section 4.2. The example here is from Berkom project, which had the development model according to DIN EN ISO 9001. In the following, the integration in current processes is described with respect to the quality criteria mentioned above. Here, the flexibility and customizability of the processes are especially important. As an accompanying process, user-centered

quality engineering spans over the entire development process steps, starting from the conception phase via specification to operational support.

- ***Preparation of Order***

  While preparing the order, the project manager worked out the basic features as well as the QA measures derived with the customer. This also includes the measures of user-centered quality engineering. Thus possibly integrate the end-user in the project development.

  Meetings with the customer, the advantages of user-centered quality management were introduced and discussed. This was to help the customer over the new method and integrate the requirement analysis for measures with user-centered quality management.

- ***Conception***

  In this phase, project planning and all further planning documents have to be created. It is necessary to detail the technical requirements of the customer, work out the technical solutions and put in specification document. The requirements from the user's point of view concerning usability, acceptance and quality-of-service analysis aspects have to be described at this point.

- ***Specification***

  Regarding the measures within the scope of a user-centered quality engineering, the individually examinations of usability, acceptance and quality-of-service analysis are described here. It is possible to include user within the scope of the specification and to carry out examinations within the scope of user-centered quality engineering.

- ***Realization & Test***

  In this phase, measures for user-centered quality engineering can help for selection of product components and solutions, and can contribute to an early product optimization from the customer's point of view. Here prototypes are evaluated or components are compared to make use of the best. Measures are taken from independently of testing activities such as functional tests, system integration tests, load tests, etc.

- ***Acceptance***

  Measures for user-centered quality engineering are not important.

- ***Support for Market Introduction***

  In this phase, measures of user-centered quality engineering are to test the product's suitability for the market. Here, usability, acceptance and quality-of-service analysis provide indicators for potential improvements. If errors are found, they should be further examine in additional usability tests. Moreover the important information about support of marketing and distribution is collected in this phase.

- ***Operational Support***

  During operation, it is interested in the questions about frequency of use and user satisfaction. This information is the basis for decisions on how to improve further.

- ***Conclusion and Archiving***

  In this phase, the experiences gained in the process of project from the user-centered quality engineering have to be evaluated, and relevant results, data and so on have to be archived according to data protection regulations.

4.4 Establishing Quality Procedures in Software Development [AND02]

Given the fact that Web projects are always short of time, it is essential to introduce a project-integrated quality management from the outset of the project. The quality objectives and criteria must be defined at the outset of the process, in co-operation with project management, since it can be broken down to the lowest production level of a software development. This introduction is based on the project developed in Dresdner Bank AG, which is an e-commerce based real-time accounting system. Dresdner Bank integrated quality management into the process flow for the realization of this major project. By defining quality objectives, establishing quality assurance measures and supporting the individual sub-projects, the quality targets have been achieved. One of quality management's targets within the scope of the overall project is to establish an acceptance procedure which ensures that quality is "built" into the process flow already during the development process.

Establishing a QM Procedure

In order to set up QM procedures in a major project, suitable approaches and methods have to been chosen from a selection of model approaches and software architectural models. These were then integrated in the project in the form of guidelines. The following are the procedures:

- *Defining* the phases of the development procedure

  - *Analytical and design phase*

    The analytical and design was performed on a UML (unified modelling language). A procedural guideline planning analysis was set up for this phase.

Examples and templates were used to indicate ways of achieving individual project results.

- *Implementation Phase*

In this phase, programming took place in Java, using the "Visual Age for Java" (VAJ) development environment. Development testing was carried out using the "JUnit" tool. An implementation guideline describing the general approach for the development test was established for this phase. These guidelines include Java programming conventions, guidelines for program documentation and guidelines for the repository incorporated in VAJ.

- Test Phase

Due to the testing approach adopted by the bank, the test phase was split into three stages: functional testing, application testing and connectivity/integration testing.

- *Definition* of Quality Management Integrated in the Process Flow

To establish a project-integrated QM process, the role of quality management within the project has to be defined from the outset.In the Bank project. The QM was to establish quality assurance measures and to provide the necessary support for these measures. It is important to note that quality management does not assume the role of policeman, but rather of an authority on quality planning, quality control and quality checking. The role of quality management in the individual project stages comprises the selection and determination of result types. The first results from each stage were used by the QM to evaluate whether the result types were useful or not. QM incorporates future review as early as in

the definition of the analytical and design phase to ensure that gaps in the result types are recognized and improved.

- *Co-operation* with other Project tasks

  QM in this procedure is to co-operate with those responsible and with project controlling on the definition of individual projects. This is facilitated by separating the responsibilities: Project controlling is responsible for monitoring and management of project planning and the production process in terms of deadlines, expenditures, budget and resources. QM is responsible for the functional acceptance of production installments, sub-projects and releases in terms of content and for planning and monitoring the quality assurance measures which need to be implemented.

Integration Quality Management in Software development processes

From a QA perspective, it should be possible to review technical feasibility and cost-benefit aspects during initial project phases. Performance, security and feasibility in terms of maintenance are important quality features in this context. In the software development process, acceptance of result types by QA must be transparent to everyone involved and responsible. Sets of requirements that apply throughout the production process need to be defined. In the Bank project, QA chose the following results as test items:

- *Analytical model:* Business entity model
- *Application documentation*: Complete description of functional features is required to achieve the milestones.

- *Development model:* Describes the lower level of classes, methods and attributes that must be implemented.
- *Development documentation:* Describes the design and aspects relevant to implementation.
- *Java code:* Java code and procedures; software documentation extracted using Javadoc.
- *Development test:* Documentation (generated with Javadoc) of test cases and testing using Junit.
- *Description of test cases* that are relevant for the functional features.

Based on the above references, acceptance protocols for chosen groups of result types are drawn up to represent result types from this acceptance process.

Figure 4.3 shows the process flow on how the QA integrated in the software development phases in Dredner Bank's project.

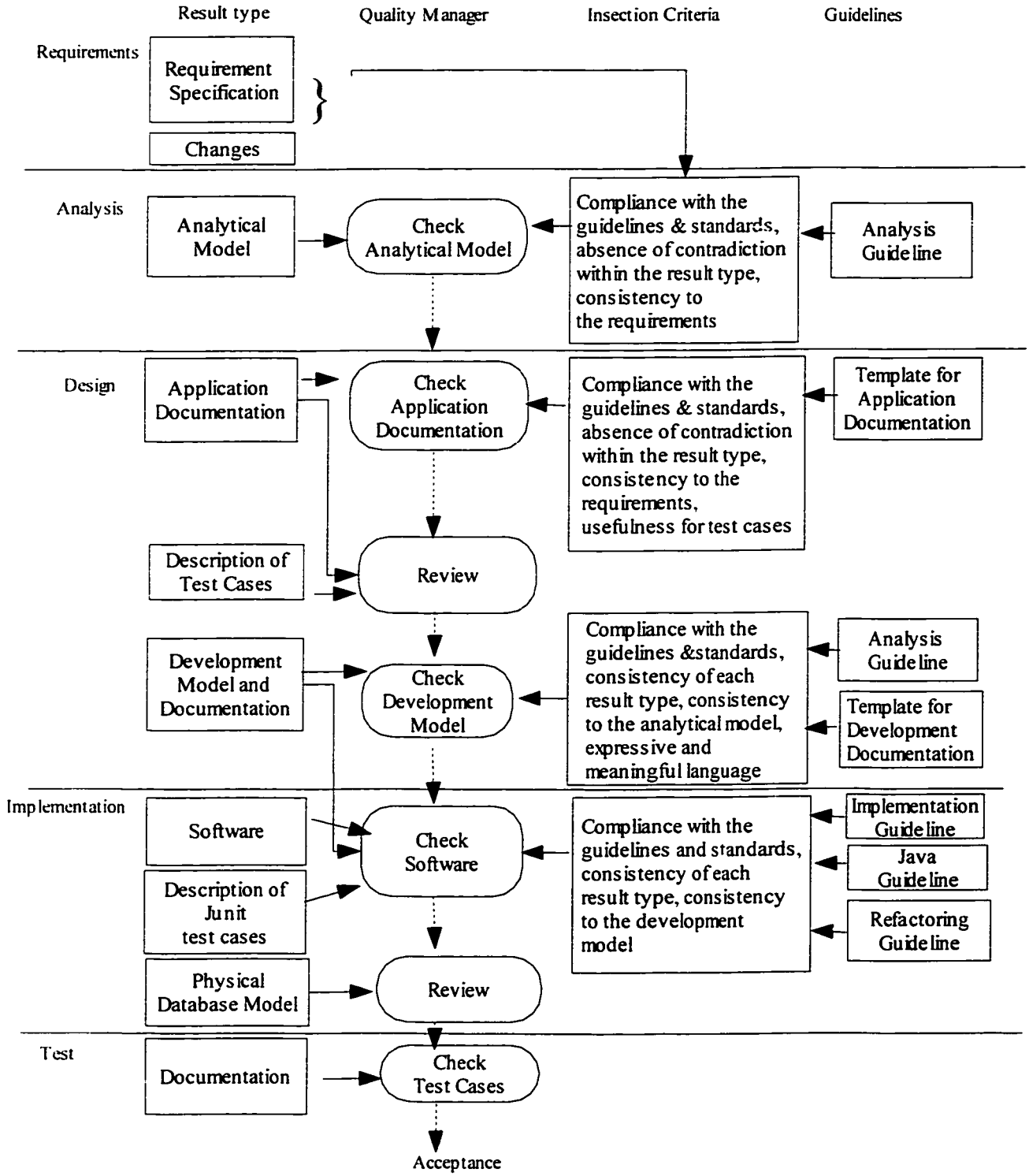

Figure 4.3 Acceptance procedure in the process flow

4.5 Web Project Risk Management [STE02]

With the rapid growth of Internet, many companies expect to merge their services to the Web to increase the speed to the market. This rush to the Web has created huge pressures in the Web development and e-business. Since the customers' needs are always in the first place. The advent of e-commerce has increased the direct visibility of application functionality to the customers no matter where they are. Customers can be other businesses, internal users or individual end-customers. Any mistakes can no longer be easily hidden or glossed over with excuses and the adverse impact materializes in terms of time, cost and damage to the business. Poor service of a Web product will make the customer run away and turn to another company (maybe the competitors). How can we develop a product in a short time limits also have a good quality product? This rises the risk control issue in the Web development. Web development and e-business in general have to identify and assess these risks. This assessment and management must be performed during development because any inadequacies are published worldwide, and to be found in the Internet will damage any business.

QA and testing can be used as a risk management technique during Web e-commerce development. QA and testing can be defined during the identification and assessment of risks. The definition and assessment of risks must be an integral part of the projects throughout the development life cycle.

4.5.1 Risk and Management Overview

We can define the risk in a general way:

Risks are things that may happen. Risk management is a technique to stop something getting worse and to help to stabilize the incident in readiness for recovery. Finally, it seeks to stop the incident from happening again.

Testing is a viable risk management technique. It is applied to minimizing the business and delivery risks in e-commerce projects. Like the testing in business process, if testing is to be used, it needs to design the testing cases to identify the potential risks. Consequently, first, we need to know what the risk is, the cause of the risk, where it is most likely to be found and the symptoms. In a word, to identify the risks. Let us see an example list of risks gained from e-commerce project development, which may be a start point to use risk management technique:

- *Poor communication:* Unavailability of appropriate resource and environment crisis management. Loss of confidence in the project and its delivered outcomes.
- *Inadequate server capacity:* Symptom poor performance/response. Transactions abandoned or time-out. Loss of existing customer base.
- *Over-complex site design:* Difficult to navigate across the site. Transactions abandoned. Few repeat user hits. Loss of existing customer base.
- *Site content breaches regulations/statutes:* Complaints by regulator, customers, and competitors. Fines or rework costs incurred.

- *Inadequate security:* Loss or malicious amendment of data. Unauthorized publication of personal data. Loss of market reputation and consumer confidence.
- *Inadequate audit trail:* Fraud. Attract unfavorable regulatory interest. Loss of reputation and consumer confidence.
- *Inefficient code:* Poor performance. Higher transaction costs.
- *Unreliable data:* Data mismatches preventing customer transaction e.g. registration, purchase, information requests. Multiple or nil information being dispatched to the customer e.g. multiple investment statements to the same address, information sent to the wrong address or recipient.
- *No business recovery process/facility:* The company is unable to transact during a major systems failure or cannot recover quickly from it. Lost customer transactions. Regulatory/statutory penalties incurred. Loss of reputation and consumer confidence.

Here, we don't discuss how to identify the risks in detail. Just mention that there are many methods for identifying risks. For example, metrics from similar projects, current project data such as the rate and completeness of planned testing, or the tools like Problem Perfect, etc.

4.5.2 QA and Testing Technique in Risk Management

QA and testing can be used to manage the risks in a Web project. By the planned tests, it can be against the risks and risk management identified. It treats the risks and risk management functions identified during the risk assessment process as subsets of the

business and functional requirements for the project. The impact of each risk must be factored into the planned tests by the prioritization and dependency.

Prioritization of the Risks

Test priorities can be used for the testers to design the test plans and strategies. Prioritization provides the focus for the application of QA and testing to ensure that appropriate test coverage is achieved. Test prioritization is not standalone. It should be considered with risk assessment a requirement of the project to prepare the test strategy and supporting plan. Following is an example of test prioritization from a Web financial application. We can see how it links with the risks identified in last section:

Priority 1: Screen navigation
- All screens are navigable from end to end, forward and back
- All field tabbing works as intended.
- Intended levels of access security proven.

Priority 2: Accuracy of displays.
- All calculations are correct
- All fields offer full display without truncation of results.
- All date processing is accurate.
- Numeric, alphabetic and alphanumeric fields have full validation including checks for special characters and appropriate error handling.
- Protected fields demonstrate protection/security/confidentiality.
- Zero spelling errors.

Priority 3: Performance
- Scroll speed meets requirements
- Table look-up speed meets requirements
- Screen navigation speed meets requirements
- Links and data transfer meet volume and speed requirements

Priority 4: Support process
- All help screens fulfil requirements
- Call center processes and service levels in place and proven
- Problem and audit support procedures in place and proven
- Business/disaster recovery in place and tested

The priorities used in this example follow a simple logic: Risks with screen navigation set at priority 1 on the basis that if the web site cannot be navigated customers are unable to use the site. Risks of accuracy of display set to priority 2 because if the customer provided incorrect information that means customers will make decision on wrong data, damage of reputation. Performance is in the third priority. If poor performance proved to be the main cause of brief visit, it may result to loss the business opportunity. Finally, priority 4 is located to line with the risks of support processes. Poor support processes are likely to result in a loss of customer retention.

Since the QA and testing is not possible to eliminate all risks, the QA and testing strategy should include the information like test objectives, test environment, approaches, a bibliograph of related project documents. This information is essential to successfully testing for any project. It ensures that detailed test planning is designed to address clear objectives and that responsibilities, timing and dependencies are clearly understood by the whole project team. Due to project time or resource constraints, it may not be possible to produce a comprehensive test plan. But prioritizing testing against the known risks will improve the project's chances of delivering an application with acceptable test coverage and clearly be understood, proven levels of functionality and performance. The risks identified in last section could generally be applied to most client-server projects, including the Web projects. Following table summarized the testing that could be applied to minimize the risks identified.

Poor Communication	1. Involve all relevant personnel (this usually extends beyond the recognized project team). 2. Test accuracy and clarity of displays. Record defects on project test log. 3. Walkthrough (role play if appropriate)support processes including manual workarounds and other contingencies. Record defects on project test log. 4. Record and publish risk assessment, test strategy, test plans & other appropriate project documentation. 5. Record and publish test results including: tests planned vs. tests successfully completed, defects reported, defects fixed and defects outstanding. 6. Record and publish issues, risks and management thereof. 7. Share learning from failure across projects.
Inadequate Server Capacity	1. Involve all relevant personnel. 2. Check the impact of other planned developments that may be hosted from the same server. 3. Check capacity forecasts (or perform for the first time). Plan against best post-implementation projections. 4. Confirm capacity availability and time-scales. 5. Test or simulate capacity scenarios using appropriate automated test tools. This simulation must include transaction loading as well as typical number of concurrent users. Record all defects on project test log. 6. Test, data transfer, reporting and backup and restore scenarios. Record all defects to test log. 7. Test speed of online navigation with a range of users and transactions. Perform at different times of day/week. Simulate navigation at peak periods of day or system cycle. 8. Ensure that third-party contracts are reviewed and amended to ensure that any new requirements identified are incorporated.
Over-complex Design	1. This should include real users representing a range of skill levels from novice through to expert to exercise end-to-end system navigation (include manual process walkthrough where necessary). 2. Employ critical-design reviews if time constraints permit. Focus upon areas of key process and transactional complexity. 3. Utilize code reviews especially for complex or high-value/volume transactional code. 4. All proposed changes to be managed via the project change management process.

Content Breaches Regulations	1. Legal and regulatory expertise needs to be factored into the project and testing from the first day. 2. Ensure that legal and regulatory expertise is applied to a screen-by-screen walkthrough of all content. This includes help text, guidance notes or illustration. 3. Legal and regulatory expertise must also apply diligent scrutiny to any supporting literature processes and audit trails. 4. All defects to be recorded on the project test log.
Inadequate Security	1. Ensure appropriate security expertise is available to the project. 2. Examine known security risks that have been experienced in other projects and factor these into testing. 3. Test access-level entry, end-to-end navigation, encryption, invalid inputs, destructive testing, error handling, malicious penetration e.g. virus protection, external and internal access, backup/restore, processes (including reporting alerts and access denial), documentation/administration/input/output controls and business recovery.
Inadequate Audit Trail	1. Ensure appropriate audit expertise is available. 2. Ensure that the input, validation, processing and output of information can be tracked (including at archive and business recovery.) 3. Ensure that audit information cannot be changed or deleted by unauthorized or untracked transaction. 4. Test that any regulatory audit requirements are fulfilled. 5. Ensure that all appropriate audit processes are in place and work.
Inefficient Code	1. Apply code reviews. Focus upon the most-used code paths. 2. Utilize code-automated analysis tools if these are available. 3. Record all defects on the project test log.
Unreliable	1. Build Test data to match the business and transaction scenarios set out in the requirements and risk assessment documents. 2. Any manual input during testing should be a mix of valid and invalid data to test validation and error handling. 3. Consider the need for any data cleanup prior to implementation and ensure that process and plans are in place to achieve this.

No Business Recovery	1. Ensure that the scope of recovery and support processes is clearly defined and documented. 2. General support processes including day-to-day call center and IT activities are walked through as part of any testing. 3. Check that incident management and escalation procedures work. 4. Ensure that an up to date business recovery plan is in place and test it. (Check that key business recovery test triggers e.g. major organizational change, major system change, elapsed time period or key personnel changes, are clearly understood and in place.)

Table 4.2: Summary of risks and QA testing activities.

4.6 Summary

In traditional software development processes, end-user is the last one who sees the project. It leaves questions relating to acceptability, usability, and quality of service unanswered. Since these quality factors are towards perspective of customers. By applying user-centered quality engineering, the end-users' view integrates into each development phase. Thus obviously, it increases efficiency and end-user satisfaction. The projects that done in Berkom which integrated the user-centered quality engineering into the development processes proved that the succeeded in designing a manageable process and fulfilling the quality requirement. The experience gained tells that the productivity and efficiency of the projects were increased and the required time reduced. Also very important is user's satisfaction increased.

In this chapter, we introduced the quality assurance in Web application project development processes. The use of user-centered quality engineering in Web project led to a higher customer satisfaction. By integrating the quality management in the development processes, the quality objectives and criteria which defined at the outset of the process, in co-operation with project management, can be broken down to the lowest

production level of a software development project. Moreover, the QA and testing knowledge or techniques employed to test client-server technology can be used to manage risk on a Web application system.

Chapter 5

Quality Model of Web Applications

5.1 Introduction

There are many books and other published material that present a lot of information on what to do and what to avoid, when designing, developing or maintaining a Web application. Lynch and Horton [LYN99] discuss typography on the Web, dealing with issues like alignment, capitalization, typefaces, etc. Although very educational and useful, this knowledge is not sufficient. In order to improve the quality of a Web site, we have to decide which quality assessment principles to apply, how to apply them, and when to apply.

This is a crucial decision in a Web application activity.

Due to the increasing growth of Web sites and Web Applications, developers and evaluators have interesting challenges not only from the development but also from the quality assurance point of view. As we know, the quality assurance was and is one of the challenging processes in Software Engineering as well as for the Web engineering, as a new discipline [MUR01]. Particularly, regarding the quality perspective, a clear definition and management of functional and non-functional requirements in order to specify, measure, control, and improve the produced Web sites and Web applications are largely needed. As Web sites have grown both in interaction and functionality, they

changed from just being static, document-oriented pages to dynamic application-oriented pages with at least the complexity of traditional software applications. This makes the evaluation and ultimately the quality assurance more challenging. There exist many design guidelines, heuristics, and metrics for the evaluation of Web sites and Web Applications. It makes more sense user-centered quality assurance in Web application design and evaluation. After all, most people who use the Web are more concerned with its end-to-end behavior.

As we see in Chapter 2, the principle technologies, which are enabling Web applications, are computer networking and telecommunications, client-server computing, multimedia and hypermedia. It can be said that Web application is a system which final scope is the satisfaction of the end user. The latter communicates with the virtual seller of the e-commerce shop, using a human-computer interface (HCI). Such interface is user-centered and should be effective for the user [AVO00]. Since the end user interacts with a computer, the quality of the interface should follow the same principles as software quality. In brief, the quality of Web application systems is related to system quality, quality of the HCI and quality of the services offered. Given the fact that all interactions with the end user are accomplished through the human-computer interface, the quality of Web application systems can be addressed in similar terms as software quality.

In Web infrastructure, there are many components working together to provide Web services. Object-oriented technology is widely used in developing Web applications. To measure the quality of Web applications, the OO measurement approach should be taken

into consideration. Some OO metrics has been introduced in Chapter 3. In this chapter, we discuss some metrics from the characteristics of Web Applications point of view.

In **Section 5.2**, we propose a quality framework for entity types grounded in the ISO 9126 quality model. Before specifying and defining a measure, it is necessary to know what attribute of what entity type it will measure. In **Section 5.3**, we discuss the quality factors of Web applications from a user-centered view. Given the relation of the aforementioned quality factors (functionality, reliability, usability, and efficiency) to the characteristics of the Web application systems, emphasis is on the effort to provide a good justification why the specific quality factors play such an important role in user satisfaction. Moreover, the correlation among each of the four quality factors and the Web application system characteristics is discussed in **Section 5.4**. In **Section 5.5**, we gives a catalogue of Web metrics, which used to support different quality assurance processes such as non-function requirement specification, quality testing definition, etc., both in the development and maintenance phases. It supports the quality model that discusses in this chapter. **Section 5.6** presents a summary of this chapter.

5.2 A Quality Framework

Quality is a property of a Web site defined in terms of a system of attributes, like consistency of background colors or average download time. When performing quality assessments, the term *quality* must be well understood and precisely defined. ISO 8402 gives the following definition:

"The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs."

According to the ISO 9126 standard [ISO91], software quality consists of six quality factors, which are *functionality, reliability, usability, efficiency, maintainability and portability*. These factors can be further analyzed in specific characteristics. Concerning Web application systems, quite often, consider the quality factor of usability as the most significant factor of software quality [SHN]. Usability issues on Web are the subjects of Paolini's research [PAO98][GAR98], according to which, usability for Web applications should focus on structure and navigation, and inspection appears to be the most suitable method. However, usability is not the only factor involved in Web application quality. The quality factors of functionality, reliability and efficiency also contribute to the user satisfaction.

Software quality is a complex, highly context-dependent concept. The goal of software quality measurement is to predict the level of quality of the software entity, and/or to monitor the improvement of the quality during the software development process. Different software quality requirements have been modeled by decomposing it into various quality factors, quality criteria and quality metrics in a hierarchical way. In Software Quality Factor-Criteria-Metrics Framework ([IEEE Std 1061-1992, 1993]) the technique is defined to build a quality model according to the organization goals and management requirements. The model's hierarchical structure is built by decomposing of every quality requirement in quality attributes (factors) from the management and user-

oriented views, by decomposition of each factor into software-oriented attributes from the technical personnel views. Software metrics of the characteristics of each criteria is identified and associated to the established criteria and factors.

A quality model may involve a lot of interdependent attributes and has to take into account the particular purpose of analysis for which quality is being modeled. Attributes of a Web site may include a very large list of properties, possibly at different levels of detail, including usability, efficiency, reliability, maintainability, and complexity.

Like any modeling activity, to define a quality model is a creative task for which it is not possible, in general, to give a precise list of steps to be followed. However a general method can tackle the problem based on the Goal, Question, Metrics (GQM) approach outlined first by Basili and Weiss [BAS94] and then often adopted in Software Engineering. The GQM approach can be followed on any analysis that requires data collection. Quality assessment is such an activity since the developers need to acquire data about the site to determine its quality level.

In the Web Engineering as in the Software Engineering field, the integral evaluation of attributes of different entities is not easy. It is difficult to consider all the characteristics and desirable attributes of a process, resource, product (like a Web site or a Web application), or product in use. Quality framework, models and methods allow the evaluators to specify systematically those characteristics and attributes. So as a first step, it is important to define a quality framework that serves as a guide in the classification

process of entities, models and associated metrics. Figure 5.1 shows a relationship among the quality factors in consideration of entities that can be in the quality assurance process adapted from ISO/IEC 9126-1. From these quality factors, the attributes and metrics can be derived.

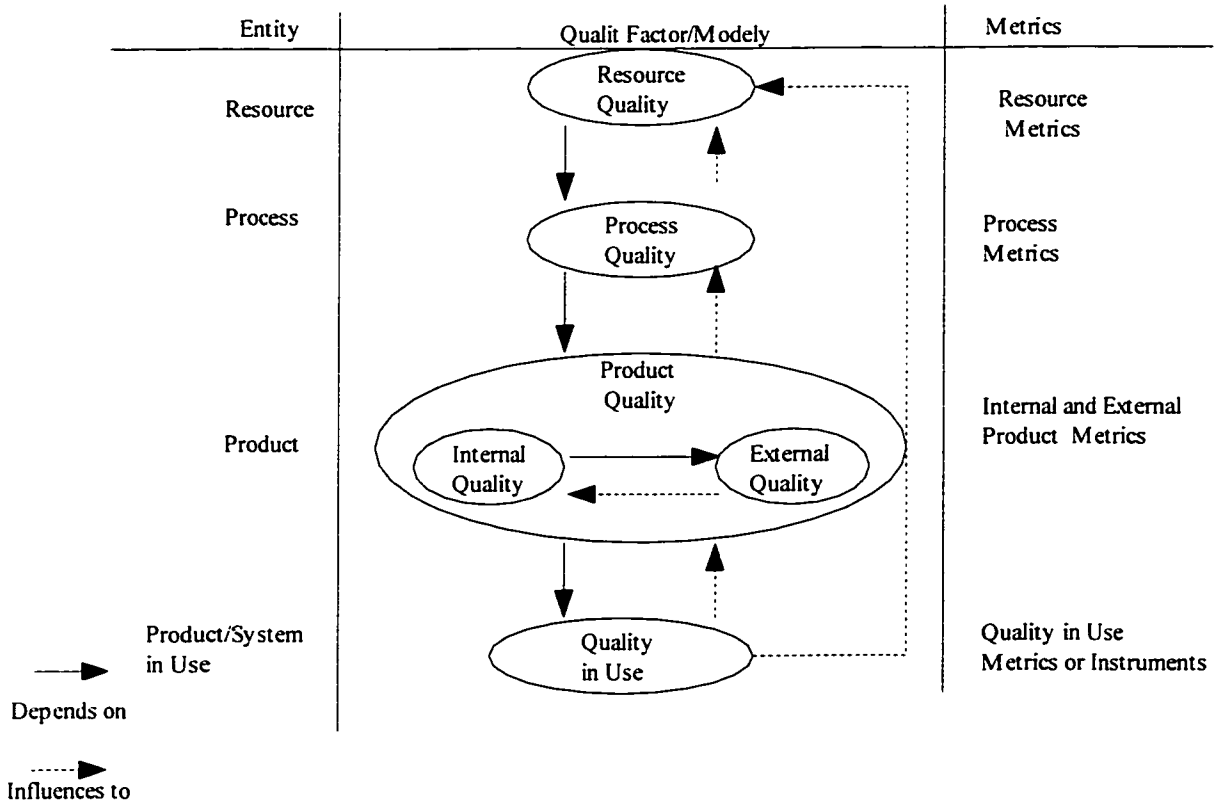

Figure 5.1: Quality framework by different entity type and potential quality models toward Web Engineering fields (from ISO/IEC 9126-1).

We implicitly observe, from the ISO/IEC 9126-1 quality model framework [ISO01], that each quality factor (e.g., product quality) belongs to an entity of the empirical domain. Because an entity can only be measured through its attributes, it is necessary to define measures of entity's attributes in order to be able to analyze the measurement data.

From this framework, we have following factors: *Quality of Resource*, *Quality of Process*, *Quality of Product*, and *Quality in Use*. In Figure 5.1, we can see that resource quality potentially contributes to improve process quality, and process quality influences the product quality, and this in turn, influences the quality in use. The evaluation of the quality in use can provide feedback for improving a product, and the evaluation of a product can give feedback for improving the process quality.

By means of *Quality of Resource* factor, a quality model to measure human or technology resources can be specified, which can influence the quality of processes. By means of the *Quality of Process* factor, we can specify a quality model to measure different aspects of a process. Using a model for the *Quality of Product*, we can model the product quality in consideration of the internal and external quality of the product. Last, by means of the *Quality in Use*, the users' perceptions and reaction interacting with the real product in specific scenarios of use is measured, considering the specific user profiles.

For instance, for the *Quality of Product* factor, the ISO 9126-1 quality model standard prescribes six well-known characteristics as well as a set of sub-characteristics for each one, which we will see in section 5.3. The hierarchical model can be specified as a tree compounded of characteristics, sub-characteristics and attributes. Let us consider the attribute *Orphan Page Count*, which measures having no return link to the site where are included in. *Orphan Page Count* attribute can be measured as a direct metric on the absolute scale. A possible indirect metric is: X = *OrphanPageCount / PageCount*. Does

this metric relate to a characteristic or a sub-characteristic? By ultimately empirical studies, we can draw this attribute as being associated to the *Reliability* factor.

Table 5.1 gives the summary of the attributes of different entity types in quality of Web applications using ISO 9126 quality model.

Entity	Sub-entity	Attribute	Definition
Resource	Personnel	Productivity	Definied as the quotient between the size of the produced oupput and the required input as effect[1]. For instance, the LOC produced per person days.
	Method/Tool	Method/Tool Usage Level	Defined as the level of use of a given method (or tool) in a Web or software project.
Process	Authoring	Interlinking Effort	Defined as the estimated elapsed time taken to interlink Web site pages.
	Testing	Link-testing Effort	Defined as the estimated elapsed time taken to test all links in a Web site or Web applications.
		Coding Faults Count	Defined as the number of faults found in code testing.
Product	Program	Code Length	Defined as the number of lines of code in a program (here, a distinction whether commented lines of code or not can be made).
	WebApp	Program Types count	Defined as the number of different programming technologies used to build programs in a Web application. For instance, JavaScript, CGI scripts, Java applets, ActiveX, etc.
	Page	Page Media Count	Defined as the number of different types of media used in a page.
Product/System in Use	Web Apps in Use	Task Completion Time	Defined as the elapsed time a user takes in completing a previously established task. We can obtain the average elased time for a user's type and compare it with the one an expert user had taken.
		User Success Rate (or Task Completeness Level)	Defining this rate as the percentage of tasks that users complete correctly. It measures users' ability to complete tasks.
		Task Completeness Efficiency	Defined as the quotient between the completeness level and the average completion time.

Table 5.1:Summary of attributes for different entity types using ISO9126 quality model

5.3 Quality Factors

Since the user interacts through a Web interface, it is obvious that Web application quality is related to the quality of the Web pages and the services that are provided to the end user. It is argued that the quality of Web application systems is related to four quality factors, which are *functionality, reliability, usability*, and *efficiency*. It is worth to mention that some of the characteristics of Web application systems are related to more than one of the above quality factors.

Functionality [KIT96]

Functionality refers to a set of functions and specified properties that satisfy stated or implied needs. Its sub-characteristics are suitability, accuracy, inter-operability, and security. Based on the definition, it is obvious that the quality of functionality can be related to the basic characteristics of Web application systems. The name of the Web site and the time needed to interact with the site's Web pages create the first impression to the user, given the fact that the user expects direct access to the Web site and navigability through the Web pages. Navigability, pleasant interface, compatibility with all kinds of browsers, multilinguality and provision of accurate information also play an important role.

For instance, in an e-commerce application, the provision of electronic shopping cart, where the user can drop and store the merchandise while he continues to navigate, is also a functional mechanism. Additionally, electronic shopping lists enable the user to create a list for future shopping, thus saving time and effort. Another important facility for the

user is the ability to find the right information at the right time. The availability of a search engine service and the creation of shopping categories can aid in reducing search time, but in order to search with efficiency, one needs an operable search engine and a functional site map.

Another basic functional characteristic example of Web applications is the procedure of payment. There are various methods of payment, such as digital currency, electronic credit card and electronic check payment. In all the above methods of payment, a very important parameter is security. The reversibility of user's actions, the existence in every stage of the transaction - of a clear exit and the confirmation, by e-mail, that the transaction has been completed are important e-commerce characteristics related to functionality.

Navigation in Web application as an e-commerce shop is similar to a walk in a real shop, where the user interacts with the seller (e-commerce system interface) and requires answers to questions. The provision of frequently asked questions (FAQs), or means of direct access to the e-commerce shop (telephone number, fax, email) should be included in a functional e-commerce system. A functional help service provides all kinds of information and instructions for a helpful navigation. Moreover, the identification of the user every time he/she visits the Web site, as well as the provision of a district server for frequent users are functions that ensure user satisfaction.

Reliability

The quality factor of reliability [FEN97] refers to a set of attributes that bear on the capability of software to maintain its performance level, under stated conditions, for a stated period of time. Sub-characteristics of reliability are *maturity*, *fault tolerance* and *recoverability*. The reliability, as far as Web application systems are concerned, is related to the accuracy of the information (text, images, multimedia) provided about products and services, as well as the consistency of the services (shopping list, shopping cart, searching). A Web application system is reliable when it restores user transactions, even in the case of a system failure.

The basic characteristic of Web application systems related to reliability is security of electronic transactions. Five blocks of security have been identified [FEN97], as far as Internet transactions are concerned. They are confidentiality, authentication, access control, data integrity and user's accountability. For this purpose, means like digital certificates and the Secure Socket Layer (SSL) have been created and their role is to guarantee the security of transactions. For example, using cryptographic method ensures the reliability of Web application systems and meant to guarantee security of transactions, even in the case of system failure. Another important characteristic of Web application, which should be provided to the user, is privacy of personal information. Certain users may want to limit the number of detailed personal information (such as buying habits or financial resources) that they are required to provide to a Web system in order to complete a transaction. Others may allow the disclosure of personal information, only if they have access to the collected information, or may want to maintain a personal

record and analysis of what personal information has been collected. A reliable Web application system should provide the possibility of such actions.

Usability

Usability is defined as a set of attributes that bear on the effort needed for the use and on the individual assessment of such use by a stated or implied set of users. According to ISO 9126, the sub-characteristics of usability are *understandability*, *learn-ability* and *operability*. Based on the definition, it is obvious that the quality factor of usability is related to characteristics of Web applications, such as provision of accurate informative texts about products and services offered, as well as provision of thumbnails, photographs and videos presenting the services and products available. Additionally, a well-designed interface that attracts user's attention and facilitates of navigation, contribute to the usability of Web application systems. Another important characteristic related to usability is easy and simple access to the Web site. A Web site can either be accessed directly (by means of its name), or indirectly (through a Web search engine like Yahoo [Yahoo], Altavista [ALTA], Lycos [LYCO], etc.)

The effective provision of services like electronic shopping cart, electronic shopping list, site map, search engine and payment methods is of great importance to an easy to use issue of a Web application. Additionally, an inexperienced user should be able to access and use the mentioned services easily, while experienced users demand fast and easy access to the Web pages that interest them, through clear paths (e.g. not having to pass through informative pages, such as the company's history, or the company's profile). A

usable Web application system should enable the end user to adapt the Web pages to his own personal profile and needs, Consequently, applications that process user profile and adjust the interaction based on one's specific needs and preferences are desirable characteristics of Web application in e-commerce.

Given an example in e-commerce, users of the e-commerce system, just as every buyer, wishes to receive the best products and services possible, with all the advantages that a simple shop provides, like offers and special prices. An e-commerce system should be updated regularly and new products should be presented, while those that are not for sale may more be removed from catalogues. It is important that an e-commerce system provides the facility of cross selling complementary or similar products. Additionally, special forms enabling the user to accurately describe the ideal product for him are very usable for searching the appropriate product. The provision of a detailed help service (avoiding, however, unnecessarily long texts) affects greatly the usability of the system. The operability, attractiveness and understandability of all the mentioned characteristics of e-commerce systems are important to the usability.

Efficiency

The quality factor of efficiency [KIT96] refers to a set of attributes that bear on the relationship between the software's performance and the amount of resources used under stated conditions. Sub-characteristics of efficiency are time behavior and resource behavior. Based on the definition above it is argued that efficiency is also important to the quality of a Web application. A system is efficient if the user can access the relevant

Web pages promptly and easily. Navigation through the Web pages should be completed at the minimum time possible, and access to the categories of products and relevant descriptive information (text and thumbnails) should be easy. Therefore, an efficient Web system should rely on user personal profile, user preferences and other user information available.

5.4 Classification of Web Criteria (Characteristics) to Quality factors

The assessment of the quality of Web application systems is based on a set of characteristics of Web applications. We can distinct characteristics of Web application system by summarizing the different characteristics of Web application systems related to the quality factors. This set of characteristics is perceived by the users of such systems and follows the quality model. The proposed user-centered characteristics of Web application systems do not just form a set of properties that a Web application system may have or not. More than that, they form a set of hierarchical characteristics, which constitute specific quality factors according to defined weights, and eventually contribute to the overall quality of the Web application system that is under assessment.

Table 5.2 gives an example that summarize different characteristics of an e-commerce Web application system related to the quality factors according to the ISO 9126 model. It lists by how important the characteristics to the quality of such e-commerce system. The first level comprises those characteristics that are most important for the quality of Web application. The second level consists of the characteristics that are related to the services provided. The third level includes the least important characteristics.

Easy access to the Web pages	Functionality, Usability, Efficiency
Easy navigation	Functionality, Usability
Adaptation to user profile	Functionality, Usability, Efficiency
Search engine service	Functionality, Usability, Reliability
Easy exit – undo function	Functionality
Useful help service	Functionality, Usability, Efficiency
Secure and reliable transactions	Functionality, Reliability
Security protocols SET, SSL.	Reliability
Direct delivery of the products	Usability, Efficiency
Recoverability of products and services	Usability, Functionality
Legitimate Web site	Reliability
Multi-linguality	Functionality, Usability, Reliability
Provision of company profile	Functionality, Reliability
Better and direct service for the frequent user	Functionality, Usability
Alternative searching services	Functionality, Usability
Site map service	Usability
Alternative presentation of the products, using images, multimedia, etc.	Functionality, Usability
Attractive interface	Usability
Categorization of products	Usability, Efficiency
Smart agents – FAQ	Functionality
Notification by email	Functionality, Usability
Cross selling	Functionality, Usability
Form for the description	Functionality, Usability
Thanking message	Usability
Variety of color and graphics	Usability

Table 5.2. An example of characteristics of Web applications

Another type of list on the quality factors covering the important areas from a testing view. Based on different authors opinions (Hung Q. Nguyen [NGU01], Tim Van [TIM], Thomas A.Powell [POW98], etc), six main areas are identified:

- *Functionality:* Links, Forms, Cookies, Web Indexing, Dynamic Interface Components, Programming Language, Databases
- *Usability:* Navigation, Graphics, Content, General Appearance
- *Server side Interface:* Server Interface, External Interface
- *Client side Compatibility:* Platform, OS, Browsers, Settings, Preferences, Printers
- *Performance:* Connection speed, Load, Stress, Continues use
- *Security:* General Security

There are very few standards for how a Web site should be designed in order to make users experience the site as user friendly and comprehensible. Web sites encountered often show poor consideration for how users respond and act on the Web. Therefore the usability issues should be addressed separately to ensure that both the functionality and usability aspects are covered.

5.5 Quality Metrics

Starting from a quality framework, a catalogue of metrics basically allows evaluators and other stakeholders to have a consultation and reuse mechanism, which starts from a sound specification of the entity type, the attribute definition and motivation, the metric formula, criteria, and application procedures. When a metric is defined and specified, it is necessary to previously know what attribute of what entity type it will measure. Here, we introduce a repository of metrics based on the quality framework (see section 5.2) for entity types grounded in the ISO 9126-1 quality model. Particularly, among the hundreds of automated Web metrics catalogued up to now for pages and sizes, different categories were identified as *Link and Page Faults*, *Navigation*, *Information*, *Media*, *Size*, *Performance*, and *Accessibility*. The Web metrics listed here focus on the idea that the given results can be extended to any other metric belonging to the product entity type.

Orphan Page Count Metric [NIEJ]

An orphan page is a page that has no internal link to the site where is included in (or it has all internal links broken). Although it can have some external links, these links will not allow navigate inside internal pages of the site. *Orphan page count* is the number of

pages that have no internal links to the Web site where they are included in. When a visitor accesses an orphan page through an external URL, s/he is unable to navigate inside the site. This kind of page has no internal navigational functionality and its utility depends rather on its content exclusively. The *Orphan Page Count* attribute corresponds to the Reliability characteristic for the ISO 9126-1 quality model. It can give stakeholders useful information both in the development and maintenance phase indicating the absence of page links that allow smooth site navigation.

Attribute Type	External or internal, depending on the process lifecycle
Scale Type	Absolute (it is a counting)

Table 5.3: Metric characteristics

Formula:

X = #OP (Number of Orphan Pages)

X >= 0, the closer to zero the better.

Procedure:

From a starting URL (of a given site page), recursively analyze all the pages, considering exclusively those that at least an internal and not broken link. Following is the generic algorithm:

```
Preconditions
Starting from the initial URL of the Web site to analyze = URLj,
Orphan_page = 0; j: 1..Page Count
Orphan_pages (URLj): #Orphan_pages
For each page (URLj) not previously analyzed
    If ¬∃ an internal (URLji) not broken then
       #Orphan_pages = 1 + Orphan_pages (URLj+1)
    else
      #Orphan_pages = Orphan_pages (URLj+1)
    end if
end for
end
```

Broken Link Count

Broken Link Count is an indirect attribute represents the total number of broken links both internal and external to the site, not including dynamically generated pages and links. It is important to know if a broken link is internal or external to the site because a broken internal link is likely caused by carelessness or by an extreme complexity in the structure, meanwhile the other, is caused by an external and uncontrollable environment. On the other hand, this attribute does not take into account the distinction among broken links to identical URLs, so all broken links are counted.

Figure 5.2 shows the procedure that automates the Broken Link Count metric. The Web site MA tool [OLL01] implements this algorithm and stores the current and destination URLs that would lead either to the internal or external broken link. This allows analyses and corrections. According to the returned HTTP state code, a broken link will be detected by this code, likewise, depending on the returned state code other link failures and metrics can be determined.

```
Preconditions
  Starting at the initial URL of the Website to analyze URL= URLi
  #Broken_links = 0;
  #Internal_broken_links = 0;
  #External_broken_links = 0;
  j: 1..PageCount.
Broken_links (URLj) : #Broken_links
For each link (URLji) of page with URL= URLj not previously analyzed
   If (URLji) is broken then
     If (URLji) is internal then
       #Internal_broken_links = #Internal_broken_links + 1
     Else
       #External_broken_links = #External_broken_links + 1
     End if
   Else
     If (URLji) is internal then
```

```
        #Broken_links = #Internal_broken_links +
#External_broken_links
                        + Broken_links()
    End if
End for
  Return (#Broken_links)
End
```

Figure 5.2: Algorithm for automating the Broken Link Count metric for static pages.

This attribute influences the quality of Web sites. From the visitor point of view, the bigger the number of broken links is, the lesser the reliability on the site is. From the developer point of view, the distinction between external and internal links (broken or not) is relevant, as commented above.

Link Count [MEN01]

Link Count is the total number of links of static pages of a site. It can be collected automatically. We can reuse the algorithm shown in Figure 5.2 to calculate the total number of internal and external links. This can be enhanced to take into account both textual and graphic links. When just internal links are considered, the metric is called Connectivity.

Percentage of Broken Links

Using the above metrics - *broken link count*, and *link count*, the percentage is calculated by the following formula:

$$PercentageBrokenLinks = 100 * \frac{\#InternalBrokenLinks + \#ExternalBrokenLinks}{LinkCount}$$

This attribute may be considered domain independent. Besides, the metric of this attribute shows to some extent how reliable a site is (the reader can also consider the quotient between the number of external broken links and the total number of external links, likewise can be done for internal links). Considering the quality impact of the metrics we could see the importance of broken links relating to their location in the more relevant or visited pages of a site. This gives place to the definition of the *Frequency of Broken Links per Hit Pages* attribute.

Number of Different Broken Links

This metric is obtained by analyzing the distinct URLs used in the *Broken Link Count* metric. Internal and external broken links to the same resource are just counted once. It can show useful information for the maintenance phase helping in the analysis of the impact on changes. In this case, the procedure to data collection and computation as specified in Figure 5.2, has a slight change, i.e., just checking if the considered URL was visited before or not.

Percentage of Different Broken Links

Instead of the *Percentage of Broken Links* metric, the relation is established according to the non-repeated links. The percentage is calculated as following:

$$PercentageDiffBrokenLinks = 100 * \frac{\#DiffBrokenLinks}{DiffLinkCount}$$

By combing distinct metrics, useful information can be drawn. For instance, we can see what is the level of link redundancy regarding the quotient between the *Different Link Count* and the *Link Count* (either internal or external or both).

Number of Images with Alternative Text [WWWC]

Images give visual information that sometimes should be deactivated by the user for accessibility or performance reasons. For example, by disabling the browser's image feature. A Web site should provide suitable mechanisms to visualize images optionally, without loosing all transmit information.

The ALT property (in the HTML code) links an alternative text with an image (or other objects in Web pages such as applets, sounds, etc.) This contributes to the readability of the page (even more the text could be read before the image is unloaded). However, the measure of this attribute does not guarantee the quality of the alternative text. Some text can be generated automatically when editing with tools like FrontPage, among others.

Image count

This metric helps to measure the amount of provided visual information (in the same way, we can alternatively measure the *Media Count* metric, where the number of media files is considered). The existence of images in a page is checked through the IMG property that is supported by the HTML code.

Percentage of Presence of Alternative Text

This is an indirect metric. It is calculated as following:

$$PercentagePresenceAlternativeText = 100 * \frac{\#ImageswithAlternativeText}{ImageCount}$$

A more careful study on the impact of this metric it would be to observe the percentage of presence of alternative text in the images posted in the firsts levels of the tree (assuming a hierarchical structure), and/or the more visited pages of a site.

Different Image Count

This direct measure counts the non-repeated images in the site.

Percentage of Image Redundancy

The relation between the amount of different images and the image count in a site can be posed as shown in the following formula. An image repetition may be interpreted as the level of redundancy of visual information.

$$PercentageImageRedundancy = 100 * \left(1 - \frac{DifferentImageCount}{ImageCount}\right)$$

Page Count

This metric can be obtained by counting the total number of static pages of a site both HTML and SHTML. It shows the initial size of the site according to the number of documents or pages, however, more elaborated metrics to measure size and length can be provided (e.g., considering the page size attribute as well).

Average Links per Page

Average Links per Page is calculated as the quotient between the *Links Count* and *Page Count*. This metric gives information about the interconnection density. In other words, it indicates how an average page is interconnected toward destination nodes.

Page Size

The size of a (static) page is measured considering all its images, sounds, videos and textual components. For each page, the size in bytes can be obtained. The size of pages is an important issue in order to appreciate the size efficiency as we posed in the following metrics.

Quick Access Pages

The download time (T) is related with the size of a page (τ) and the speed in the established connection line(c).

$$T\mathit{Download} = f(\tau, c)$$

This time is directly proportional to the page size and inversely proportional to the speed of a given connection line. A function may be created in order to classify pages as quick or slow access pages, according to a minimum threshold of time (e.g. 10 seconds) for a given speed of a connection line. Nielsen [NIE00] gives an example of recommend page sizes depending on the speed of communication lines.

$$g(T_{Download}) = \begin{cases} \mathit{QuickAccess} & T_{Download} < T_{max} \\ \mathit{SlowAccess} & T_{Download} \geq T_{max} \end{cases}$$

This is a simple way to measure the performance or predict it at design time, however, it does not reflect the actual or perceived performance, as the reader might surmise.

5.6 Summary

In this chapter, we discussed quality model for assessing the quality of Web application that based on the ISO 9126 quality model framework. It analysis the quality factors emphasising on the user-centered factors of functionality, usability, reliability and efficiency. A repository of metrics is presented.

Chapter 6

Tools Support for Quality Assessment in Web Applications

6.1 Introduction

Web technologies change at an extremely rapid speed, and Web sites and Web application development follows the speed. Driven by market pressure Web sites contents have to be updated very frequently, and redesigns of a Web site (its contents, information architecture and look and feel) occur very often. Nevertheless a constant or an improved quality level is required to generate and maintain user trust and motivation to use the site.

Using a methodology based on a quality model for the Web site can support this activity of the Web applications. A quality model specifies which properties are important for a Web site (e.g. its usability, its performance, and its visibility) and how these properties are to be determined. A Quality assessment procedure can be used both to determine the current state with respect to quality and to plan future actions. The assessment itself is based on a system analysis integrating multiple interdependent views on a software system into a coherent analysis environment. With the advent of Web applications, especially e-business applications, through We testing has become much more than a matter of customer satisfaction only. It is also a business need.

The measuring activity is likely to take much time and effort. In this case, the automatic tools are used for analyzing Web sites. There are already many automatic tools for Web

metric collecting and analyzing in the market right now. The automatic tools for analysis, being systematic and mostly automatic, are crucial parts in a methodology based on quality models for assuring constant quality levels.

Usability is an important aspect of software products, especially in Web projects. Usability of a product can be tested from two different perspectives: easy-of-use and quality-in-use. Quite often the scope is limited to the first perspective. The ease or comfort during usage is mainly determined by characteristics of the software product itself, such as interface. SUMI does precisely that: it allows quantification of the end users' experience with the software and it encourages the tester to focus on issues that the end users have difficult with.

In this chapter, we introduce Web measurement tools for quality assessment.

Section 6.2 introduces quality assessments from software system view. **Section 6.3** introduces SUMI method in usability measurement. In **Section 6.4**, we introduce a concept to measure the performance of Web site. **Section 6.5** describes how to use the functional size to measure Web application maintenance and cost. In **Section 6.6**, we discuss the test coverage of a Web application using Combinator Metrics Method. **Section 6.7** introduces the topic of using automate testing tools in Web quality model.

6.2 Software Quality Assessment in Software System [FRA02]

One of the major goals in today's software development is to produce high quality software. Quality is one of the most important factors for the commercial success of a

software system. The concept of quality can be different with respect to the view on the system. The end user of the system is primarily interested in the usability and user-friendliness of the system. On the other hand, the developer is interested in the typical engineering goals such as maintainability, efficiency and portability. Thus quality can be divided into external quality to cover the end users' interests and internal quality to cover the engineers' interests.

To assessment a system with architecture and design is more focus on internal quality. A system with low internal quality typically causes high maintenance risks, high maintenance costs and high training effort. Consequently, to ensure the internal quality is to concentrate on the quality aspects from the very first stage of the software development process.

Following is the definition of quality assessment from ISO 9241:

"An action of applying specific documented assessment criteria to a specific software[...] product for the purpose of determining acceptance or release of the software product."

This gives two main requires before applying quality assessment:

- Quality requirements to be used for the quality assessment.
- Artefacts to be assessed.

Quality Requirements

Quality requirements define the criteria used for the quality assessment. Two major criteria can be identified:

- *Individual quality requirements* (I-QR) describe subjective desires.
- *State-of-the-art quality requirements* (Sota-QR) describe needs for which consensus is possible in the corresponding community.

A prerequisite for a quality assessment is the set of quality requirements to be checked in the assessment. The quality requirements to be checked depend on the type of artefacts to be assessed. For example, even if the CASE tool is used to support continues development, in many cases the software engineers change the design while implementing the program skeletons that were generated by the CASE tool according to their initial design. One major goal of a quality assessment is the identification of deviations between quality requirements and the examined source code.

A quality assessment is based on different views of the system. There can be four levels that extracted from the system hierarchical structure: *system*, *architecture*, *design,* and *code*.

- *System* is the overview of the whole system. Typical individual aspects for this level might be described in development guidelines containing general requirements.
- *Architecture*: The highest abstraction of the software system is assessed. In source code this level usually is mapped onto a corresponding package structure. Typical

individual quality requirements for this level might be documented in technical papers.

- *Design*: In this level, the designs of the components of the architecture are assessed. Typical individual aspects for this level might be stored in technical designs.
- *Code*: In this level, considers the single lines implementing of a special design. This level is often explained in projects by programmer handbooks.

Quality Assessment Process

The following quality assessment process is an extension of the ISO 14598 in some degree, which is concerned with he evaluation of software product quality.

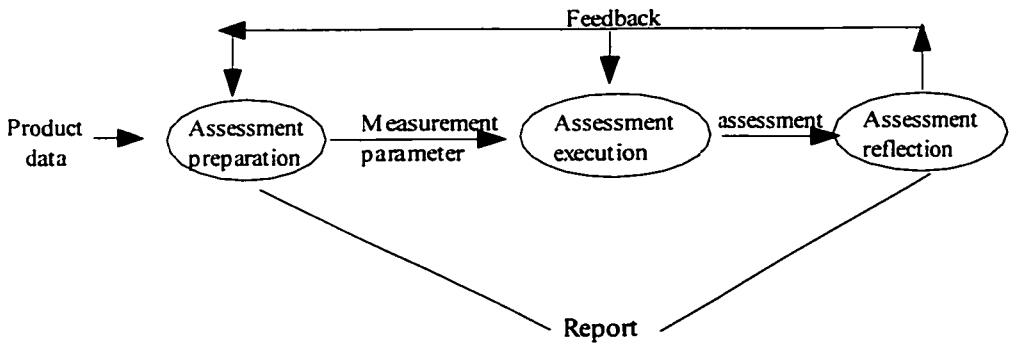

Figure 6.1: Quality Assessment Process

- The *assessment preparation* corresponds to the three ISO-sub-processes analysis, specification and design. The process output is called evaluation plan. The goal is to set the goals of the assessment and to prepare the tool environment.

- In the *assessment execution* step, the assessment itself is performed and the report to the project is prepared. This step is carried out with the help of the software analysis workbench.
- In *assessment reflection* step, the results are presented to the project and further actions are planned. The results of the quality assessment can be used as input for further assessments.

Software Analysis Workbench

In the above quality assessment process, the software analysis workbench is used to help the analysis. The core of this workbench consists of a relational database system, which containing all relevant structural and measurement data of the software system being assessed. The data in database is derived from parser interfaces, e.g. Wind River's SNiFF+ programming environment. On the top of the workbench, there are a number of analysis tools that address specific software analysis issues. Table 6.1 shows the tools used in this workbench and their description.

ArchitectureChecker	It provides a language for describing the architecture of a system in terms of a set of layered architecture models. Based on the models the ArchitectureChecker checks whether there are illegal relationships in the source code.
QueryTool	It manages the definition and execution of powerful queries and their representation according to the underlying data model. This allows the definition of problem-specific sets of queries that describe the generation of both basic as well as complex analysis views.
XrefBrowser	It provides high-level cross-referencing functionality between flexible definable abstraction levels.
MetricsInspector	It is an enhanced software metrics tool. It overcomes the typical weak points of most existing software metrics tools by providing a user with comprehensive support for browsing and filtering metrics values.
GraphVisualizer	It allows the creation of 2D and 3D graph representations of entity sets as UML-like class diagrams or inheritance trees.

Table 6.1: Example of Tools used in the software analysis workbench

6.3 Software Usability Measurement Tool: SUMI [ERI02][STD24]

Within the European ESPRIT project MUSiC, a method has been development that serves to determine the quality of a software product from a user's perspective. Software Usability Measurement Inventory (SUMI) is a questionnaire-based method that has been designed for cost-effective usage. SUMI is a solution to the recurring problem of measuring the user's perception of the usability of software. It provides a valid and reliable method for the comparison of (competing) products and different versions of the same product as well as diagnostic information for future development. SUMI is not a substitution of a full usability test. In fact, SUMI "only" measures the user's perception of usability - one aspect of usability test.

SUMI consists of a 50-item questionnaire devised in accordance with psychometric practice. Each of the statement is rated with "agree", "undecided" or "disagree". The following lists the sample of the kind of questions:

- This software responds too slowly to inputs.
- I would recommend this software to my colleagues.
- The instructions and prompts are helpful.
- I sometimes wonder if I am using the right command.
- The way that system information is presented is clear and understandable.

SUMI is intended to be administered to a sample of users who have had some experience in using the software to be evaluated. In order to use SUMI reliably, a minimum of ten users is recommended based on statistical theory. Usability scores are calculated based on

the answers given and statistical concepts. Before SUMI can be performed, it needs a working version of the software system exist. This working version can also be a prototype or a test release.

One of the most important aspects of SUMI is the development of the standardization database, which now consists of usability profiles of over 2000 different kinds of applications. Basically any kind of application can be evaluated using SUMI as long as it has user input through keyboard or pointing device, display on screen and evaluating a product or series of products using SUMI.

SUMI gives a global usability figure and additional readings on following five sub-scales:

- *Efficiency*: degree to which the user can achieve the goals of his interaction with the product in a direct and timely manner.
- *Affect*: how much the product captures the user's emotional responses.
- *Helpfulness*: extent to which the product seems to assist the user.
- *Control*: degree to which the user feels that s/he is setting the pace
- *Learnability*: ease with which a user can get started and learn new features of the product.

Figure 6.2 shows an example of SUMI output. It shows the scores of a test and the spreading of these scores against the average score of the reference database.

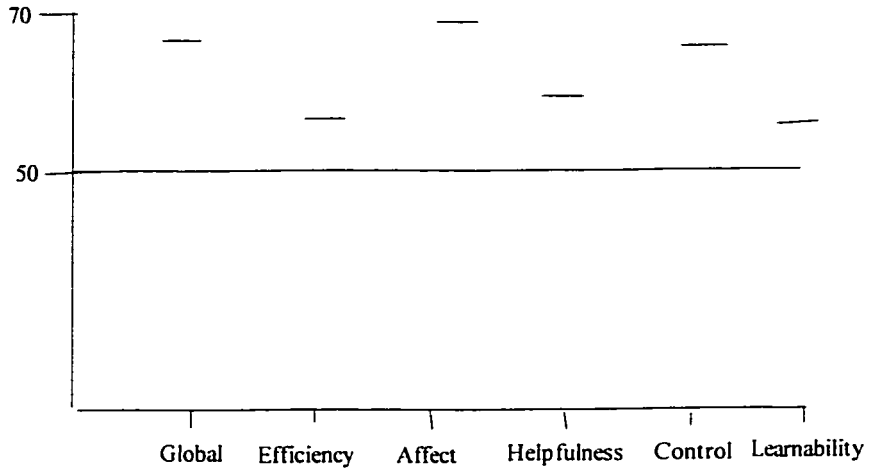

Figure 6.2: A sample profile showing SUMI scales

SUMI is the only available questionnaire for the assessment of software usability, which has been developed, validated and standardizes on a European-wide basis. The SUMI sub-scales are referenced in international ISO standards on usability and software product quality. Product evaluation with SUMI provides a clear and objective measurement of the user's view of the suitability of software.

6.4 Measure Web Application Performance [VAL01]

Performance can be described simply as the raw speed of the applications in terms of a single user. It's fairly easy to measure performance. We can use the application being tested or we can design an automatic benchmark and observe the original speed of the application against it. Then we can changes to the software or hardware and determine if the execution time has improved. This is a very simple approach, but on Web environment, because of the complexity of Web infrastructure, many components could affect the quality of Web applications. Valeria and Emiliano [VAL01] gives a performance study for quality Web services on distributed architecture. They consider the

following three main classes of load in Web environment: a Web site may provide one or a mix combination of the following models:

- **Static Web Services Model.** Requests for HTML pages with some embedded objects. Typically, this model has a low impact on Web server components. Only requests for very large files are disk and network bound.
- **Dynamic Web Services Model.** Requests for HTML pages, where objects are dynamically generated through Web and back-end server interactions. Typically, these requests are CPU and/or disk bound.
- **Secure Web Services Model.** Requests for a dynamic page over a secure connection. Typically, these services are CPU bound because of overheads to setup a secure connection and to execute cryptography algorithms.

A Web site may provide one or a mix combination of the above models. The number of consecutive Web pages a user requests from the Web site (page requests per session) follows the inverse Gaussian distribute [JEP99]. The user's think time is modeled through a Pareto distribution [JEP99][BAR99]. The number of embedded objects per page request including the base HTML page is also obtained from a Pareto distribution [JEP99]. Web files typically show extremely high variability in size. The function that models the distribution of the object size requested to the Web site varies according to the object type. For HTML objects, the size is obtained from a hybrid function where the body follows a log normal distribution, while the tail is given by a heavy-tailed Pareto distribution. The size of distribution of embedded objects is obtained from the log normal distribution [BAR99].

Table 6.2 summarizes the parameters' value that used in the so- called static model.

Pages per session	Inverse Gaussian	μ=3.86, λ= 9.46
User think time	Pareto	α = 1.4, κ = 1
Objects per page	Pareto	α= 1.245, κ=2
HTML object size	Lognormal Pareto	μ= 7.630, σ= 1.001 α= 1, κ = 10240
Embedded object size	Lognormal	μ= 8.215, σ= 1.46

Table 6.2: Static Model

A dynamic request includes all overheads of a static request and overheads due to back-end server computation to generate the dynamic objects. They consider three classes of requests to the back-end nodes that have different service times and occurrence probability. Light, middle-intensive and intensive requests are characterized by an exponential service time on back-end nodes with mean equal to 16, 46 and 150 msec, respectively. The three classes represent 10%, 85% and 5% of all dynamic requests, respectively. These last parameters are extrapolated by the log file traces of two real e-commerce sites. Table 6.3 summarizes the parameters of so called dynamic model.

Light Intensive	16 mses	0.1
Medium Intensive	46 mses	0.85
Intensive	150 mses	0.05

Table 6.3. Dynamic Model.

Secure transactions between clients and Web services involve the SSL protocol. The model includes main CPU and transmission overheads due to SSL interactions, such as key material negotiation, server authentication, and encryption and decryption of key material and Web information. The CPU service time consists of encryption of server

secret key with a public key encryption algorithm such as RSA, computation of Message Authentication Code through a hash function such as MD5 or SHA, and data encryption through a symmetric key algorithm, such as DES or Triple-DES. Most CPU overhead is caused by data encryption (for large size files) and public key encryption algorithm (RSA algorithm), that is required at least once for each client session, when the client has to authenticate the server. The transmission overhead is due to the server certificate (2048 bytes) sent by the server to the client, the server hello and close message (73 bytes), and the SSL record header (about 29 bytes per record). Table 6.4 summarizes the throughput of the encryption algorithm used in the secure model.

RSA(256bit)	38.5
Triple DES	46886
MD5	331034

Table 6.4. Secure Model

To evaluate a Web site, normally it needs to mix the three models together. In Web Service Application level, the performance is typically measured as the K-percentile of the page delay that must be less than Y seconds. Typical measures are 90- or 95-percentile of the requests that must have a delay at the server less than 2-4 seconds, while 7-8 seconds of response time (including also the time for the address lookup phase and network transmission delay) are considered acceptable at the client side.

In the design of a Web site it is necessary to know the maximum number of clients per second that the system could serve with the requested services. The *break point* of the Web site is recommended to this value. To analyze when the network connection of the Web site to Internet starts to become a bottleneck, use the *peak throughput* that is, the

maximum Web system throughput measured in Mbytes per second (Mbps). Over certain peaks, it is necessary to pass from a locally to a geographically distributed Web system.

6.5 Web Functional Size Measurement

As empirically demonstrated complex Web application structures can increase maintenance costs considerably. Even worse, for continually changing Web applications (considering their innate evolving nature), the cost can be even higher. In order to diminish risks, the maintenance cost can be predicted through a Web-centered Functional Points measure [ABR01].

Function Point Analysis (FPA) [IFP99] has become one of the most popular software functional sizing metrics. The functional point metric is an approach for the early evaluation of software characteristics such as size, productivity and cost. In addition, it provides a normalization factor that allows products to be compared independently of implementation technologies.

Before functional size measure, it is necessary to determine the estimation model for the Web application quality evaluation. The functional size measurement process of a Web application is accomplished by mapping *Function Point* concepts to the quality model. The mapping rules are based on the standard FPA defined in the IFPUG count Practice Manual [IFP99]. This mapping is conducted in the following steps:

- *Define* the application boundary and the measuring scope
- *Identify* the data and transactional function types

- *Determine* the complexity of each identified function. The logical functions are mapped for low, average and high complexity levels in accordance with IFPUG tables.
- *Map* the complexity levels into values. The total value results in unadjusted OO-method Function Points for Web (OomFP-Web) [ABR01].
- *Adjust* OomFP-Web value according to the general characteristics of the Web application taking into account the nonfunctional requirement specification.

Defining the Application Boundary and the Measuring Scope

The application boundary indicates the border between the project being measured, and the external applications or user domain. It defines what is external and what is internal to the application according to the user's point of view. The measuring scope limits which functionality will be measured in a particular measure (the sub-set to be measured).

The scope of current Web applications varies widely: from small Web services to large enterprise applications distributed across the Internet. For instance, in a large e-commerce application can have many sub-stores that correspond to the different subsystems of a Web application. Thus, a measuring scope can include one application and its subsystems, all functions of a subsystem, some web services, or the functionality provided to a specific user, etc.

Identifying Data and Transactional Function Types

In the OO paradigm, an object is a collection of data (attributes and properties) and functional logic (methods). The data defines the state of the object and the functional logic defines the behavior of the object. In accordance to the FPA concepts, data is a logical group maintained or modified through an elementary process (method), and a transaction must have processing logic that is unique and represents the smallest unit meaningful to the user. Based on these considerations, the following functions can be identified:

- **Navigational Classes**: Select every navigational class of a navigation context as a candidate for an Internal Logical File function (ILF). In *Function Point* terms the parts of an object corresponds to the logical structure of the file concept. Optional and mandatory subgroups of files are called Record Element types (RETs). An object that is aggregated into another object constitutes a subgroup.
- **Method**: It is a unit of functional logic contained within an object. The select every method of a navigational class as a candidate for an External Input function (EI).
- **Context Relationship**: The relationships between navigational classes express inherits or aggregation relationships in the Object Model. This kind of relationship provides the ability to access a unique navigational context as the set of data, retrieved from the navigational classes (ILFs). A context relationship expresses navigation to another context (the target context). In this kind of relationship, two behaviors can occur: a) when the primary user input is navigating without change the state of application, no information crosses the

boundary; b) when a hyperlink that sends a parameter that is used to search could be an example of an EQ. The current applications require high user interaction where the navigation may influence the content of a navigational context. That is, the hyperlink follows the rules required for an EQ: there is an input side (the parameter) and there is an output side the results of the search. In this case the output side is dynamic and changes.

- **Navigational Context**: Select each navigational context as a candidate for an External Output (EO) or External Inquiry (EQ) function. An EO or EQ must reference at least one internal logical file and / or one external interface file. Both an internal logical file and an external interface file must be a logical group of related information. Each navigational context has a main class (manager class) from where navigation starts, and others classes (complementary classes) to giving additional information to instances of the manager class.

Establishing the Complexity for Data and Transactional Function Types

As in FPA, key to developing repeatable predictor counts is a well-defined set of counting rules. Thus, to each navigational class a functional complexity is assigned according to the number of Data Elements Types (DETs) and the number of Record Element Types (RETs). The proposed counting rules for DETs and RETs identification of a navigational class are:

- Count a DET for each attribute of class
- If exists a filter associated to the class, count a DET by each attribute that appears in the formula of the filter

- Count a RET for the navigational class
- Count a RET for each context or contextual dependence relationship that has the source the class that is being considered. Each one of these relations is a group of identifiable data by the user (target navigational class)
- If exists a filter associated to the class, count a RET by each navigational class that is referenced in the formula of the filter. Whenever this supposes the reference to a class that has still not been counted

The transaction functions (EOs and EQs) will be identified from each navigational context (exploration or sequence). If in some of these contexts the attributes of the participant navigation classes indicate some calculation (derived attributes, complex filters, etc.), it will be considered an EO. In the opposite case it is considered an EQ.

To each navigational context a functional complexity is assigned based on the number of Data Element Types (DETs) that processes the elementary process, and the number of File Type Referenced (FTRs) to which such a process accedes. Each EO or EQ function has two parts: the input side where the user provides the information and the output side where the result is presented to the user. We are considering only the DETs unique that cross both sides. Then, the proposed counting rules for DETs and FTRs identification of a navigational context are the following:

- Count a DET for each attribute of all navigational classes that appears in the context.

- Count a DET for each attribute (context, link, filter or role) of a context relationship.
- Count a DET for each service that can be performed.
- Count a DET for the traversal order of the elements of the context.
- Count a FTR for each navigational class that appears in the context.
- Count a FTR for each navigational class referenced in the definition of a filter attribute (in a context relationship).
- Count a FTR for each class referenced in the population filter formula.

Finally, each service of a navigational class will be considered an External Input (EI). Nevertheless, if the same service can be executed from different contexts by the same agent, it is counted only once. The arguments of a certain service will be those that are defined for that service in the Object Model. As well as previously mentioned, to each identified service a functional complexity based on the number of DETs and FTRs is assigned. The dynamics for a service and the rules for determining the functional complexity are described in [PAS01].

Using the counting rules above described, we have been able to predict the partial size of a Web application, i.e., the size of each navigational map according to the functionality offered to the agents of the system (functionality provided by the Navigation Model). Integrating this partial result with the functional size obtained from applying the OOmFP metric (functionality provided by the others OO-Method conceptual models) the functional size of whole Web application is obtained.

The Cost Prediction

The definition of a metric for size measurement is a first step in developing a model that accurately predicts Web development costs. Software size is a normalizing reference measurement that can be successfully used as the base for a range of planning and control indicators. Including studies for the software production (function points per person-month), financial (cost analysis) [COST], software quality (defects per function points) [COW99], and to measure the functionality and productivity of Web applications .

Most organizations use measures based on function points to assist in estimating project costs, effort and duration. Now, we will see how the function points measure can be used in conjunction with financial indicators. These indicators can include value of the software, project costs, IT costs, and maintenance costs. The followings are examples of financial indicators:

- **Asset Value**: (Cost /FP) x Portfolio Size
- **Project Costs**: Total Project Costs /FP
- **Enterprise Costs**: Total IT Costs /FP
- **Maintenance Costs**: Total Cost of Maintenance / FP (including overheads)

The asset value places a value on the installed Web application. The cost can be replacement cost, total cost, etc. An organization has to establish which of these it will use. The project cost is the unit cost per Function Point that includes all costs the development project occurs, and the total project cost is derived from:

Total Project Cost = (work effort hours χ hourly cost) + other expenses

The enterprise cost is the cost equivalent to the enterprise productivity defined earlier. It measures the total cost to the enterprise, including overhead cost, to deliver the new or enhanced functionality to the user. Finally, the maintenance cost is the cost equivalent to the Web Application support rate. It measures the cost of application maintenance and support. The total cost of maintenance is obtained by:

Total Cost of Maintenance = (Hours of Work Effort x Hourly Costs) + other expenses

Thus, size-based measures have traditionally been the basis for estimation models to predict costs of software activities.

6.6 Web Test Coverage Measurement [THO02]

Testing means comparing the observed behavior with the expected behavior of a system. In web environment, because of the specific structure of Web system, Web testing includes testing of business processes, system testing as well as testing of different client-side and server-side software components. With the B2B (business-to-business), B2C (business-to-customer) applications merged into Web, human factors have become more and more affective on Web applications, thus Web testing has become much more than a matter of customer satisfaction only. It is further necessary to integrate factors like availability, accessibility, as well as user-centered quality factors into the Web testing strategy, since it would be shortsighted to concentrate only on how the system presents

information needed to execute business processes. Rather it would test the true behavior of the Web system from the customers' point of view.

The preparation of tests for a Web-based system involves considerably more effort than conventional testing. Spending significant investments for the Web testing is reasonable only if we can reuse the test investments while the Web system evolves. Metric collection in Web testing is an issue, since in different phase there are different measure focuses. Some tools introduced to collect metrics. This is where Combinatory Metrics starts to play a crucial role for the Web application.

Combinatory Metrics

Combinatory Metrics is an extension of comprehensive Quality Function Deployment (QFD). It is used when we cannot measure directly. For example, we can provide cost & revenue projections, but it is difficult to measure the goals and objectives that the Web-based service must satisfy to reach the financial targets. This is because complete specifications cannot exist. Infinite combinatory models are used to approximate such system that we cannot specify completely. QFD constitutes such an infinite combinatory model for quality that today is widely in use.

Elements of QFD

QFD allows two basic operations:

One describes how a solution approach contributes to solve a problem. The problem is stated as how to satisfy customer's needs, the solution to it is the optimum choice of

solution components that constitute the solution approach. The items such as solution approaches and problems are called *Deployment Topics*. The target deployment topics are called *Goal Topics*. The description of the solution approach is called *Solution Topics*. Such correction matrices are called *Cause-Effect Combinators*. The cause-effect relation itself is described by an approximated value that is suitable to the topics combined.

The other action describes how two Cause-Effect Combinators F * G may be applied to each other. The *Solution Topics* are the cause for the *Goal Topics*. Then it is the cause-effect combinator G, and the *Goal Topics* become F in turn. We thus can daisy chain our cause-effect combinators and thus get profiles for the solution topics down the line. Such combination chains are called Comprehensive QFD.

Test Coverage Combinators

For the Web development model we discussed in chapter 4, there are four topic levels: customer's needs, business processes, Web-based system architecture, and Web software components. Since the topic item itself changes quite often, using combinator metrics is not a significant investment compared with Web testing. Table 6.5, 6.6 show the test coverage respective to the business processes and customers' need, which use the combinator metrics in an hotel Web-base application.

	Test Case Identifier	Short Description
Forms functionality	Case1.Information completeness	Static review and dynamic semi-automated test if everything is visible.
	Case2.Field validation	Automated test for all fields.
	Case3.Internationalization	Regression tests with change of language
Acknowledgement	Case1.Confirmation pages	Correctness of confirmations, Automated API test.
	Case2.Workflow notification	Correctness of workflow. Automated API test.
Business contacts	Case1.CRM tract and exception Handling	Browser screen scripts with various exceptions.
	Case2.E-mails received and Answered	Business process test; no software involved (except e-mail)
	Case3.Phone numbers work Internationally	Conducted once with help of international partners.
	Case4.Process owners in charge	Repeated QMS audit; business test.
	Case5.Customer information Correctness	Focus group customer data is checked with customer for correctness.
Compliance	Case1.Law compliance test	Static review and dynamic semi-automated test if everything is visible.
	Case2.Marketplace compliance Test	Business inquiry rerun after each new release.
	Case3.Geographic compliance Test	Test if locations are understandably correct.

Table 6.5: Application tests for the business processes

	Test Case Identifier	Short Description
Navigation	Case1. Ergonomic tests	Lab tests that let users process their requests and watches them
	Case2. User associations	What users associate with each of the items shown
	Case3. Process navigation	Automated Test; Counts the number of clicks needed for a process.
	Case4. Usage statistics	Automated data collection. Should match the New Lanchester Theory predictions.
Business	Case1. Hit Counter	Automated data collection. Should be the target customers.
	Case2. Order statistics	Automated data collection. Should match the New Lanchester Theory predictions.
Satisfaction	Case1. Expectation inquiry	Short 1-page satisfaction inquiry
	Case2. 4 hour memory	Group of testers been asked what they remember after 4 hours.
	Case3. 36 hour memory	Group of testers been asked what they remember after 36 hours.

Table 6.6: Acceptance Tests for the Web-service

6.7 Automatic Testing Tools in Web Quality Model

Automatic Web testing tools can play an important role in the definition and usage of quality models because they are necessarily adopt objective metrics only, systematic and error-free, and much more cost-effective than any other manual method. From the quality modeling view, there are several flavors of such tools:

- Tools for accessibility testing and repair. e.g. bobby, a-prompt, 508accessibility suite, site valet, accverify, lift, etc. the metrics implemented by these tools corresponds (more or less accurately) to official accessibility criteria.
- Tools for usability testing (e.g. lift) are based on usability guidelines, like the ones that can be distilled from Nielsen [NIE00][NIEJ]. Brajnik [BRA00] gives an analysis of which usability-related attributes can be dealt-with by automatic tools.
- Tools for performance testing (e.g. Topaz)
- Tools for security testing (e.g. Webcpo)
- Tools for analyzing Web servers logs
- Tools for classifying a Web site after learning the classification criteria from other Web sites (e.g. Webtango).

Obviously, automatic tools cannot assess all types of quality properties. In particular, anything that requires interpretation (e.g. usage of natural and concise language) or that requires assessment of relevance (e.g. alt text of an image is equivalent to the image itself) will be out of reach. Nevertheless these tools can highlight a number of issues that have to be later inspected by humans and can avoid highlighting them when there is

reasonably certainty that the issue is not a problem. For example, a non-empty alt can contain placeholder text (like the filename of the image); it is reasonably simple to write heuristic programs that can detect such a pattern and flag such an issue only when appropriate -- and be able not to flag it if the alt contains other text.

Therefore if the quality model includes attributes that are easy treatment by automatic tools, the quality assessment problem can be solved by appropriately configuring the tool, running it to acquire relevant data, and then weighing the data found by the tool according to the importance criteria defined in the model. Such an activity, being based on systematic and objective analysis of the web site, is at once both economically feasible and relatively error-free.

As pointed out by Brajnik [BRG01] the issue of validity of the metrics adopted in a quality model arises when metrics are computed by automated tools and when metrics start dealing with more interesting properties, like assessing accessibility or usability. In these cases a tool may lead incorrect answers (i.e. incorrect values associated to attributes included in the quality model). This may happen either because the tool found some ***false positive*** (i.e. an issue has been reported where there is none) or because of ***false negatives*** (i.e. the tool was unable to detect a problem). Methods like the ones proposed by Brajnik can be applied, and they can limit the consequences of this problem. In general only relatively simple quality models will be based entirely on automatic tools. In the vast majority of the cases, quality assessment will be based also on human inspection and human judgment. However, the contribution that automatic Web testing tools can bring to

quality assessment of web sites is significant: low cost and superficial analyses can be performed automatically. Only thereafter, if needed, a more in-depth and accurate human analysis can be performed. In this way, productivity of systems will be enhanced.

6.8 Summary

In this chapter, a quality assessment was described and quality measurement method that used in Web application quality measurement was introduced. Although the quality assessment itself is mainly based on a static analysis of the source code it is used for checking the quality of a software system on four levels namely the system, the architecture, the design and the source code. The software analysis workbench is applied in the projects to allow the practical quality assessments of software systems. A questionnaire such as SUMI represents the end results of a lot of research effort. SUMI is testing techniques that can be applied to start usability testing or when limited resources for usability testing are available. Web function points can be a useful approach for analyzing, assessing and potentially restructuring a Web application in the operative phase. An example on Web system performance assessment was given, which considered Web sites with a mix of static, dynamic and secure requests.

Chapter 7

Conclusions

7.1 Conclusion

Because of the increasing size, complexity, and quick time-to-market cycles for Web applications, together with unsystematic use of Web engineering approaches, there are increasing concerns about the ways in which the Web applications are developed, maintained, and their quality delivered. As Baskerville said [BAS01], "*In the Internet speed development developers often defer certain aspects of product quality such as scalability and maintainability during the original development. If the software doesn't catch the market, no time was lost adding quality.*" There are compelling reasons for a systematic and disciplined use of engineering methods and tools for developing, assessing and maintaining Web sites and applications. Different types of methods and techniques are developed for evaluating quality and functionality such as testing, inspection, inquiry, simulation, etc., in the current practice. But many of them provide only partial solutions for the actual state of Web applications, because they separately focus either on nonfunctional requirements or on functional requirements. In fact, flexible and integral solutions are needed.

Software development and the evolution of the Web are interdependent in many ways [JEN02]. Even though in a Web environment, system development is often done "on the fly" and without any formal methodology, larger and especially strategic applications

require a sound methodological base that supports processes and assures quality. New requirements for Web applications generate a need for systems development research, for example, in the areas of method application, time-to-market, quality aspects, etc.

The Internet itself can act to enable for new ways of system development. For example, the open source software and free software development, like the development of free operating system Linux and open Web server Apache. This is based on a model of software development that can be generalized.

The Internet is a fast moving environment and new technologies are available almost daily. This gives a need that any methodology supporting the processes must evolve with the Web.

All of the above essentially is hoping to improve the quality of the Web application systems as a whole. As time factor in Internet-oriented systems has top priority, within the resulting frame there are different solution strategies that aim at reducing the development time and at the same time not falling below a certain quality standard.

In this paper, we discuss the quality factors of Web applications, starting from specifying a quality model based on a user-centered approach. In order to contribute to the comprehension and selection process that metrics can be useful to support different quality assurance processes such as non-functional requirement definition and specification, metrics understanding and selection, quality testing definition, either in the

inspection, development or maintenance phases. We survey the current technologies of integrating QA into Web application system development processes and the features they have. We also describe techniques for measuring quality aspects of Web applications, based on a user-centered approach.

It is worth to remark that effective and efficient quality assessment and restructuring processes require both methodological and technological support. There are many issues we need to address in the future. Such as a deeper research on how heuristics can be mapped to metrics for quality models of Web applications, a more robust adjusted quality model for the Web applications.

Bibliography

[NEL00] Nelson, M., "*Fast Is No Longer Fast Enough*", Information Week, June 2000, http://www.informationweek.com/789/web.htm

[ZWA96] Zwass, V., "*Electronic commerce: Structures and issues*", International Journal of Electronic Commerce, Vol. 1, No. 1, 1996.

[PRE97] PRESSMAN, R. S, "*Software Engineering: A Practitioner's approach*", 4th edition, McGraw-Hill Book Company, 1997.

[GIL92] Gillies, A.C., "*Software Quality, Theory and management*", Chapman Hall Computing Series, London, UK, 1992.

[FEN97] Fenton, N. and Pleeger, S.L., "*Software Metrics: A Rigorous and Practical Approach*", 2nd Ed. PWS Publishing Company, 1997.

[MUR01] Murugesan S., Deshpande Y., Hansen S., Ginige A. *"Web Engineering: A New Discipline for Development of Web-based Systems"*. LNCS 2016 of Springer-Verlag. 2001

[GHO98] Ghosh S., "*Making business sense of the Internet*", Harvard Business Review, March-April, 1998.

[JIM] Jim C., Conallen Inc. *"Whit paper: Modeling Web Application Design with UML"* http://techweekly.com/essay546.htm

[GRE02] Greg Barish, "*Building Scalable and High-Performance Java™ Web Applications Using J2EE™ Technology*". Addison Wesley Professional, 2002.ISBN:0-201-72956-3

[POW98] Powell, Thomas A., "*Web Site Development, Beyond Web Page Design*", 1998; Prentice Hall; ISBN 0136509207

[HO97] Ho, J. "*Evaluating the World Wide Web: A global study of commercial sites*" Journal of Computer-Mediated Communication, 3(1). 1997 [Online] http://www.ascusc.org/jcmc/vol3/issue1/ho.html

[DIR02] Dirk M., Begona L., Rob Van D.P.K., Andreas G.(Editors), "*Software Quality and Software Testing in Internet Times*", Springer Verlag, June 2002.

[CON86] Conte, S. D., H. E. Dunsmore, and V. Y. Shen, "*Software Engineering Metrics and Models*", Menlo Park, Calif Benjamin/Cummings, 1986.

[BER98] Bertrand Meyer, "*The role of object-oriented metrics*", in *Computer* (IEEE), vol. 31, no. 11, November 1998, pages 123-125.

[GRA87] Grady, R. B. and D. R. Caswell, "*Software Metrics: Establishing a Company-Wide Program*", Engle- wood Cliffs, N. J. Prentice-Hall, 1987

[TEG92] Tegarden, D., Sheetz, S., Monarchi, D, "*Effectiveness of Traditional Software Metrics for Object-Oriented Systems*", Proceedings: 25th Hawaii International Confernce on System Sciences, January 1992, pp. 359-368.

[LIN98] Linda H. Rosenberg, "*Applying and Interpreting Object Oriented Metrics*", presented at the Software Technology Conference, Utah, April 1998.
http://satc.gsfc.nasa.gov/support/STC_APR98/apply_oo/apply_oo.html

[NOR00] Norman Fenton, "*Software Metrics For Control And Quality Assurance CourseOverview*",2000
http://www.dcs.qmul.ac.uk/~norman/Courses/mod_903/slides/slides_2000/all_slides_2000_blue

[ALE98] Al Ehrbar , "*Software measures*", John Wiley & Sons, 1998

[NOE00] Norman E Fenton and Martin Neil, "*Software Metrics: Roadmap*", 2000
http://www.agena.co.uk/postcript_papers/metrics_roadmap.pdf

[MCC94] McCabe, "*Object Oriented Tool User's Instructions*", McCabe & Associates, 1994

[JOH92] Johann Eder, Certi Kappel, and Michael Klagenfurt, "*Coupling and cohesion in object-oriented systems*", 1992
http://citeseer.nj.nec.com/cache/papers/cs/2479/ftp:zSzzSzftp.ifs.uni-linz.ac.atzSzpubzSzpublicationszSz1993zSz0293.pdf/eder92coupling.pdf

[LIO96] Lionel C. Briand, John W. Daly, and Jurgen wust, "*A unified framework for coupling measurement in object-oriented systems*", 1996
http://www.iese.fraunhofer.de/network/ISERN/pub/technical_reports/isern-96-14.pdf

[LIO97] Lionel Briand, Prem Devanbu, Walcelio Melo, "*An investigation into coupling measures for C++*", 1997
http://citeseer.nj.nec.com/cache/papers/cs/1336/http:zSzzSzwww.research.att.comzSz~premzSzicse97.pdf/briand97investigation.pdf

[SHY94] Shyam R. Chidamber and Chris F. Kemerer, "*A METRICS SUITE FOR OBJECT ORIENTED DESIGN*", IEEE Transitions on Software Engineering, 1994
http://www.pitt.edu/~ckemerer/clnieee.pdf

[SHY91] Shyam R. Chidamber and Chris F. Kemerer. "*Towards a Metrics Suite for Object Oriented Design*", In Proc. of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'91), pages 197--211. ACM, 1991

[BIN97] Bindu S. Gupta, "*A critique of cohesion measures in the object-oriented paradigm*", 1997
http://citeseer.nj.nec.com/cache/papers/cs/2183/ftp:zSzzSzcs.mtu.eduzSzpubzSzottzSzreportszSzmehra-thesis.pdf/gupta97critique.pdf

[LIM98] Linda M.Ott and James M. Bieman, "*Program slices as an abstraction for cohesion measurement*", 1998
ftp://ftp.cs.colostate.edu/pub/bieman/istPreprint98.pdf

[FER96] Fernando Brito e Abreu and Rita Esteves, Miguel Goulão, "*The Design of Eiffel programs: Quantitative Evaluation Using the MOOD Metrics*", Proceedings of TOOLS'96 USA, Santa Barbara, California, July 1996.

[IBR93] I.Brooks, "*Object-oriented metrics collection and evaluation with a software process*", Proc. OOPSLA '93 Workshop Processes and Metrics for Object-Oriented Software Development, Washington, D.C., 1993

[RHA98] R.Harrison, S.J. Counsell, R.V. Nithi, "*An evaluation of the MOOD set of object-oriented software metrics*", 1998
http://www.computer.org/tse/ts1998/e0491abs.htm

[FBA96] Fernando Brito e Abreu and Walcélio Melo, "*Evaluating the Impact of Object-Oriented Design on Software Quality*", 1996
http://www2.umassd.edu/SWPI/ESEG/3IntSoftMetSymp.pdf

[SDI] Sdida Benlarbi and Walcelio L. Melo, "*Polymorphism Measures for Early Risk Prediction*", 1999
http://delivery.acm.org/10.1145/310000/302652/p334-benlarbi.pdf?key1=302652&key2=8583876101&coll=portal&dl=ACM&CFID=1931617&CFTOKEN=83627676

[VIC01] Victor Laing and Charles Coleman, "*Principal Components of Orthogonal Object-Oriented Metrics*", (323-08-14), 2001
http://satc.gsfc.nasa.gov/support/OSMASAS_SEP01/Principal_Components_of_Orthogonal_Object_Oriented_Metrics.pdf

[KHA00] Khaled EI Emam, Walcelio Melo, Javam C. Machado, "*The prediction of faulty classes using object-oriented design metrics*", The journal of systems and software 56(2001) 63-75, 2000
http://www.mestradoinfo.ucb.br/Prof/wmelo/jss-oo.pdf

[EIK01] El-Emam, K., "*Object-Oriented Metrics: A Review of Theory and Practice*", 2001 ftp://ai.iit.nrc.ca/pub/iit-papers/NRC-44190.pdf

[LCB00] Lionel C. Briand, Jürgen Wüst, John W. Daly 1 , and D. Victor Porter, "*Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems*", 2000 http://www.sce.carleton.ca/faculty/briand/jss.pdf

[VRL96] Victor R.Basili, Lion C. Briand, and Walcelio L. Melo, "*A validation of object oriented metrics as quality indicators*", IEEE transactions on software engineering, Vol 22,No. 10, 1996 http://www.mestradoinfo.ucb.br/Prof/wmelo/ieeetse1996.pdf

[DAN00] Daniela Glasberg, Khaled El Emam, Walcelio Melo, Nazim Madhavji, "*Validating object-oriented design metrics on a commercial java application*", 2000 http://www.mestradoinfo.ucb.br/Prof/wmelo/NCR_1080.pdf

[CUT00] Cutter C. "*Poor Project Management - Problem of E-Projects*", October 2000, http://www.cutter.com/consortium/press/001019.html

[FLE00] FLEMING, J. "*Web Navigation: Developing the User Experience*", O'Reilly & Associates. 2000.

[MEN01] Mendes, E., Mosley, N., Counsell S. "*Web Metrics - Estimating Design and Authoring Effort*". IEEE Multimedia, January-March 2001.

[POO02] Pooley R., Senior, D., Christie, D. "*Collecting and Analyzing Web-Based Project Metrics*". IEEE Software 19(1), January/February 2002.

[OLS00] Olsina, L., "*Quantitative Methodology for Evaluation and Comparison of Web Site Quality*", Doctoral Thesis, UNLP, La Plata, Argentina. 2000.

[OLS01] Olsina L., Papa, M.F., Souto, M.E., Rossi, G. "*Providing Automated Support for the Web Quality Evaluation Methodology*", Proceed of the Fourth Workshop on Web Engineering, at the 10th International WWW Conference, HONG Kong. 2001.

[MIC02] Dirk M., Begona L., Rob Van D.P.K., Andreas G.(Editors), "*Software Quality and Software Testing in Internet Times*" – "*Designing Processes for User-oriented Quality Engineering (Michael M., Astrid D., Elmar F.)*", Springer-Verlag, ISBN 3-540-42632-9. 2002.

[AND02] Dirk M., Begona L., Rob Van D.P.K., Andreas G.(Editors), "*Software Quality and Software Testing in Internet Times*" – "*Establishing Quality Procedures for Incremental Software Development (Andreas K., Tessa D., Stephan F.)*", Springer-Verlag, ISBN 3-540-42632-9. 2002.

[STE02] Dirk M., Begona L., Rob Van D.P.K., Andreas G.,(Editors), "*Software Quality and Software Testing in Internet Times*" – "*Using QA for Risk Management in Web Projects(Steve W.)*", Springer Verlag, June 2002.

[JEN02] Dirk M., Begona L., Rob Van D.P.K., Andreas G.,(Editors), "*Software Quality and Software Testing in Internet Times*" – "*Software Development in Internet Times – Overview and Perspectives (Jens L. and Ulrich H.)* ", Springer Verlag, June 2002.

[ISO01] ISO/IEC 9126-1, "*International Standard, Software Engineering - Product Quality - Part 1: Quality Model*". 2001

[AVO00] Avouris N. "*Introduction to Human Computer Interaction*". Diavlos Publications, 2000

[ISO91] ISO. Information Technology - Evaluation of software - "*Quality characteristics and guides for their use.*" International Standard, ISO/IEC 9126: 1991

[SHN] http://www.hbuk.co.uk/ap/ijhcs/webusability/shneiderman/shneiderman.html

[PAO98] Paolini P. "*Hypermedia, the Web and Usability Issues*". Proc. of the IEEE International Conference on Multimedia Computing and Systems, Vol.1, 1998

[GAR98] Garzotto F. et al. "*A Framework for Hypermedia Design and Usability Evaluation*" Chapman &Hall, 1998.

[BAS84] Basili V.R. and Weiss D. "*A methodology for collecting valid software engineering data*". IEEE Trans. on Software Engineering, SE-10(6), 1984.

[KIT96] Kitchenham B. and Pfleeger S. "*Software Quality: The Elusive Target*". International Journal: IEEE Software, January 1996.

[Yahoo] Yahoo. http://www.yahoo.com

[ALTA] Altavista. http://www.altavista.com

[LYCO] Lycos. http://www.lycos.com

[NGU01] Nguyen Hung Q., "*Testing Applications on the Web. Test Planning for Internet-Based Systems*". Wiley, New York, ISBN 0-471-39470-X, 2001.

[TIM] Tim Van T., "*Web Testing*", http://www.csgp.com/its/articles/websitetesting.html

[NIE98] Nielsen, Jakob. "*Web Usability: Why and How*". 1998. http://www.zdnet.com/devhead/stories/articles/0,4413,2137433,00.html

[NIEJ] Nielsen, J., 1996-2001, "*The Alterbox*", http://www.useit.com/alterbox/

[OLL01] Olsina, L., Gonzalez Rodriguez,J., Lafuente,G.J.; Pastor, O.; "*Towards Automated Web Metrics*", VIII Quality Brazilian Workshop. 2001.

[WWWC] WWW Consortium, "*WAI Accessibility Guidelines: Page Authoring*". http://www.w3c.org/TR/WD-WAI-PAGEAUTH/

[NIE00] Nielsen, J., "*Designing Web Usability: The Practice of Simplicity*". New Riders Publishing. 2000

[LYN99] Lynch P. and Horton S. "*Web Style Guide*", Yale University, 1999.

[FRA02] Dirk M., Begona L., Rob Van D.P.K., Andreas G.(Editors), "*Software Quality and Software Testing in Internet Times - Software Quality Assessments for System, Architecture, Design and Code (Frank S., Clause L., Walter B.)*", Springer-Verlag, ISBN 3-540-42632-9. 2002.

[ERI02] Dirk M., Begona L., Rob Van D.P.K., Andreas G.(Editors), "*Software Quality and Software Testing in Internet Times - Low-Cost Usability Testing (Erik Van V.)*", Springer-Verlag, ISBN 3-540-42632-9. 2002.

[STD24] http://www-ist.massey.ac.nz/cphillips/159353/353Evaluationweb/std024.htm

[VAL01] Valeria C., Emiliano C, Michele C. "*A Performance Study of Distributed Architectures for the Quality of Web Services*". Proceedings of the 34th Hawaii International Conference on System Sciences. 2001.

[JEP99] J.E. Pitkow. "*Summary of WWW characterizations*". *World Wide Web*, 1999.

[BAR99] P. Barford and M.E. Crovella. "*A performance evaluation of Hyper Text Transfer Protocols*". In Proc. ACM Sigmetrics 1999.

[ABR01] ABRAH?, S. M., AND PASTOR, O. "*Estimating the Applications Functional Size from Object-Oriented Conceptual Models*". In Proceedings of IFPUG Annual Conference,Las Vegas, USA, September 2001.

[IFP99] IFPUG. "*Function Point Counting Practices Manual*", Release 4.1, International Function Points Users Group, Mequon, Wisconsin, USA, 1999.

[PAS01] PASTOR, O., ABRAHAO, S. M., "*A FPA-like Measure for Object-Oriented Systems from Conceptual Models*". In *Proceedings of 11th International Workshop on Software Measurement*, Canada, Shaker Verlag, 2001.

[COST] COST XPERT GROUP, Inc. "*Estimating Internet Development*", http://www.cutter.com/consortium/press/001019.html

[COW99] COWDEROY, A. J. C. "*Size and Quality Measures for Multimedia and Web-site Production*". In *Proceedings of* 14th International COCOMO/SCM Forum, Los Angeles, USA, 1999.

[MOR99] MORISIO, M., STAMELOS, I., SPAHOS, V., "*Measuring Functionality and Productivity in Web-based Applications: A Case Study*". *In Proceedings of METRICS'99,* Florida, USA, Nov. 1999.

[THO02] Dirk M., Begona L., Rob Van D.P.K., Andreas G.(Editors), "*Software Quality and Software Testing in Internet Times - Business-Oriented Testing in E-Commerce (Thomas F.)*", Springer-Verlag, ISBN 3-540-42632-9. 2002.

[BRA00] Brajnik, G. "*Automatic web usability evaluation: what needs to be done?*", in Proc. Human Factors and the Web, 6th Conference, Austin, June 2000, www.dimi.uniud.it/~giorgio/papers/hfweb00.html

[BRG01] Brajnik G. "*Towards valid quality models for websites,* in Proc. Human Factors and the Web",7th Conference, Madison, WI, June 2001. www.dimi.uniud.it/~giorgio/papers/hfweb01.html

[BAS01] BASKERVILLE, R., LEVINE, L., PRIES-HEJE, J.,BALAS., R., AND SLAUGHTER, S. "*How Internet Software Companies Negotiate Quality*", IEEE Computer 34(5), 2001

Useful resources:

http://www.stickyminds.com

http://www.io.com/~wazmo

http://www.mtsu.edu/~storm

http://www.pcwebopaedia.com

http://www.informationweek.com