A Quality-Driven Approach to Web Testing

Rudolf Ramler, Edgar Weippl, Mario Winterer,
Wieland Schwinger, and Josef Altmann

Software Competence Center Hagenberg GmbH, Hauptstrasse 99,
A-4232 Hagenberg, Austria
{rudolf.ramler, edgar.weippl, mario.winterer,
wieland.schwinger, josef.altmann}@scch.at
http://www.scch.at

Abstract. The appropriate fulfillment of quality requirements of Web-based systems is essential for the success on the World Wide Web. Thus, in contrast to conventional software testing, where the focus is mainly on functionality, a wide range of quality issues are of utmost importance in testing Web-based systems. In this work, we present a systematic approach to Web testing that takes the importance of quality aspects for Web projects into account. Generally, tests can be characterized in three dimensions: quality, feature, and phase. The combination of these three dimensions results in a scheme that can be used to guide the different steps throughout the test workflow in a methodically sound and systematic way. The application of the scheme in test planning and design is demonstrated for security aspects on a sample Web application.

1 Introduction

Usability, compatibility, security, performance, availability, as well as reliability are considered as key success criteria of business on the World Wide Web [7]. Consequently, non-functional aspects, i.e. quality aspects, are of foremost concern in testing Web-based systems. Since the focus of conventional software testing approaches is mainly on functionality, this marks an importance shift towards non-functional or quality aspects.

Software testing, nonetheless, is concerned with the need to assure quality and has been discussing quality aspects for decades [15, 23]. Furthermore, the development of real-time, distributed, or mission-critical systems has a long tradition in dealing with quality aspects like performance, security, or reliability. Web testing, however, cannot draw from an equal tradition and in many cases conventional testing practices cannot be directly applied to Web testing due to the special circumstances found on the World Wide Web [20]. Web projects have to cope with the dynamic, fast changing, and multidisciplinary situation on the Web as well as the high complexity and integration of applications or sites. Thus, not only a few well-defined quality aspects but a wide range of different ones have to be considered. In addition, Web projects are typically conducted with very limited resources and within a very restrictive schedule, often less than one or two month [20]. Therefore, in contrast to conventional software testing, one of the new challenges in Web testing is to provide lean measures and simple techniques to methodically assure the appropriate implementation of a manifold set of quality aspects.

Web engineering [8] has accepted this challenge. In reaction to the growing demand for testing quality aspects of Web-based systems numerous suggestions have been made addressing, however, mainly selected aspects. In [21], testing of e-business systems on the Web is described with emphasize on performance, reliability, and availability. Scalability, security, and usability are named alongside as those quality aspects that should as well be covered by testing. Another approach to performance testing of Web applications is explained in [26], whereby also the necessity for ongoing periodical maintenance and monitoring of the quality of a Web system is pointed out. Although [12] deals primarily with behavioral and structural testing of Web applications, the additional need of testing quality aspects like compatibility or security, is mentioned in this paper, too. Approaches for testing a number of different quality aspects, namely, security and privacy, performance and scalability, compatibility, as well as usability, are covered in [7] and [24].

Taking a different but related view on quality aspects, [1] outlines the importance of data quality and introduces a methodology for Web site evaluation. An extensive treatment of evaluating Web site quality can be found in [16, 17, 18]. There the authors present a methodology for assessing the quality in different phases of the lifecycle, supported by a quality requirement tree based on characteristics and attributes in accordance with the ISO/IEC 9126 standard [9]. The idea of applying the ISO/IEC 9126 standard also to Web testing is suggested in [2].

Even though there seems to be common agreement on the importance of quality aspects and how Web testing is supposed to verify the required quality level of these aspects, it is less clear what quality aspects play a crucial role in a certain project and, furthermore, how all the relevant aspects can be tested in a systematic way.

Often, the quality aspects named are used as broad categories for corresponding tests. It is common practice to talk about usability testing, compatibility testing, security testing, performance testing, and availability testing, referring to tests without exactly specifying which system features and which quality aspects are affected in particular, respectively which test methods and approaches are applied. Moreover, these different viewpoints – features, quality aspects, and test methods – are sometimes mixed up and named along likewise without distinction.

In practice, the difference between these viewpoints is not as clear as it appears in theory anyway; let alone the distinction between functional and non-functional requirements [11], which largely depends on the level of detail of the specification of the requirements. Nonetheless, it is useful to be aware of the different perspectives to ensure that Web testing follows a systematic and comprehensive approach. Therefore, concepts that support planning and design of tests for Web-based systems considering quality aspects are required.

The remainder of this paper is structured as follows. In section 2 a three-dimensional scheme for testing Web quality aspects is introduced. Section 3 explains how this scheme can be applied via two-dimensional views. By means of a sample Web application section 4 demonstrates the integration of the scheme into the test workflow. A summary and concluding remarks are given in Section 5.

2 A Scheme to Support Testing of Web Quality Aspects

To methodically address quality aspects in Web testing we developed a scheme to organize and classify tests in relation to quality aspects as well as system features, and, furthermore, in relation to a time dimension. This scheme has been derived from an approach we originally used in a case study with an industrial partner to describe and evaluate the testing efforts in Web projects.

Testing concerns various quality aspects as well as different system features. From the viewpoint of testing, quality aspects and system features are orthogonal. They can be understood as two separate dimensions of testing. While the first dimension focuses on quality aspects that are relevant for the system under test, the latter and orthogonal dimension focuses on the features of the system under test. The idea to distinguish between quality aspects and features, treating each as a separate dimension, emphasizes that the features are principally the test objects – those parts that are actually executed or analyzed in testing – while the quality aspects are rather concerned with the test goals – they define what the tests actually try to ensure.

Both dimensions are required to describe a test. Consider, for example, the following three tests: checking the correctness of an operation, revealing security violations in a critical function, or finding style incompatibilities in a graphical visualization. These tests are – on the one hand – described by their objectives, the quality goals correctness, security, and compatibility, and – on the other hand – by the objects to be tested, the particular operation, function, or feature. Testing along one of these two dimensions always includes testing of the other dimension, too. And, in the end, the result of testing should be the same in either case.

For a systematic approach, however, it is useful to distinguish between these two dimensions so it will be possible to identify all the features affecting a certain quality aspect or, vice versa, all the quality aspects affecting a certain feature. This is especially important since not all features may equally – or even not at all – affect the relevant quality aspects. Taking, for example, a simple electronic shopping application, the customer should be free to look around and browse through the offered stock; little security precautions are thereby necessary, e.g., no authentication or data encryption, unless the customer is going to purchase an item. In our scheme quality aspects and system features are therefore represented as two separate dimensions.

In addition to these dimensions we propose to consider a separate, third, dimension showing the phases in which quality aspects are tested. Testing concerns all phases of a system's lifecycle rather than a single, independent phase somewhere near the end of the project adhered to development. Thus, this dimension is useful to describe the timeframe within the testing activities take place: In the early phases of the project requirements and design considerations are subject to testing, followed by unit testing and integration testing during the implementation phase, and extending over system and acceptance testing – when the implementation has been accomplished – to regression testing and monitoring in the operation and maintenance phase.

As a result, testing can profit from valuable synergies when taking the activities over the whole lifecycle into account, e.g., by reusing tests for regression testing on the one hand but also for periodically evaluating the quality, i.e. for system monitoring, on the other hand. This is especially critical for Web systems since the underpinning technologies are subject to frequent changes. For example, whenever a new browser version is released, regression testing is required to ensure inter-operability and compatibility, even when the system itself has not changed at all. Furthermore, adding a time dimension helps to establish a general view of testing over all phases and allows a better understanding of what effects a quality aspect or a feature over time. It also makes it easier to estimate investments for testing in early phases since the possible payoff in later phases becomes clear.

Putting the pieces – the three dimensions quality, features, and phases – together, the result can be visualized as a three-dimensional cube like in Fig. 1, since each of the three dimensions represents a discrete and finite axe: quality has a limited amount of aspects, the system has a limited amount of features, and the lifecycle usually has a limited amount of phases. The tests themselves are nodes inside this cube, residing at the intersection points of quality aspects, features, and phases. Thus, the question of how a certain quality aspect corresponding to a certain feature is ensured in a certain phase leads to a clearly defined point within this three-dimensional cube.

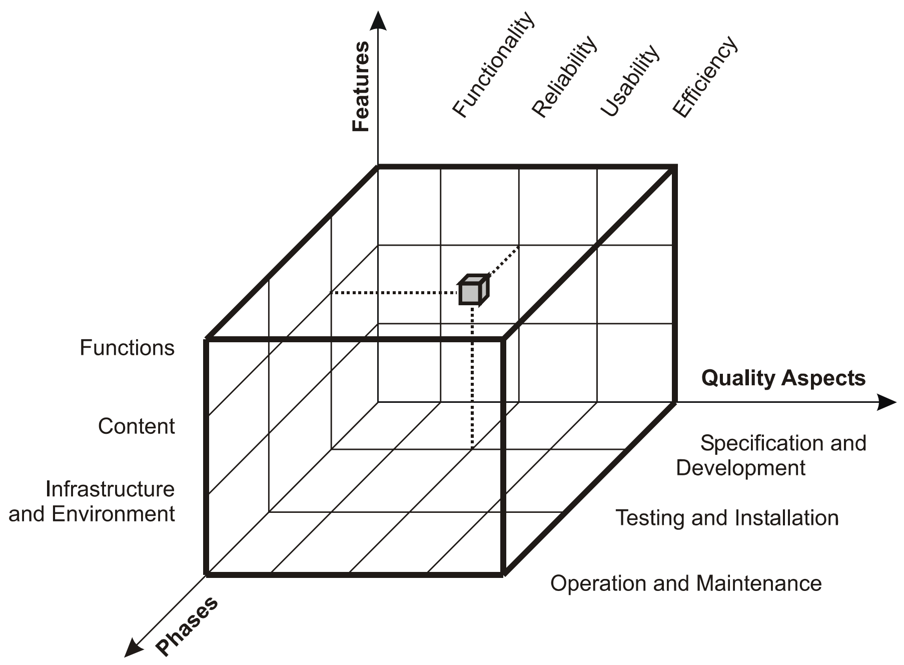

Fig. 1. The cube describes a generic scheme to organize tests for quality aspects of Web applications. The node at the intersection point of a certain quality, feature, and phase represents a test within this scheme

In the following subsections we further explain these three dimensions and present a generic classification for each of the axes to allow a preliminary general overview of the proposed scheme. In practice, however, the classification on the axes principally depends on the requirements of the system under test and its environment. Therefore, it is necessary to customize and detail the classification according to the specific situation of the project, as it will be shown for the example in section 3.

2.1 Quality Dimension

The quality dimension describes the quality aspects that are subject to testing. Traced back, the origins of quality aspects are the stakeholders' concerns. A stakeholder is anyone who has a reasonable interest in a system, for example, customers, users, developers, maintainers, managers, and the general public. From the stakeholders' primary concerns, in turn, the functional as well as the non-functional requirements are derived [4]. Since non-functional requirements relate to quality aspects, the requirements specification is one source from where a – highly project specific – classification of the quality dimension may be derived.

A further source of quality aspects is quality assurance, which provides an extensive set of quality characteristics that may serve as a more generic classification. These quality characteristics are usually part of quality models [5, 13], implemented as metrics to measure the quality of certain aspects of a system. Integrated into a quality model, these measurements are aggregated to high-level key figures, which are in turn used for general quality estimations and predictions. To encourage a common understanding of the underlying principles of the different structured quality models, the ISO/IEC 9126 international standard on quality characteristics and guidelines for their use [9] has been published. This standard provides a general purpose model which defines six broad characteristics of quality: functionality, reliability, usability, efficiency, portability, and maintainability. These six characteristics can be further broken down into measurable sub-characteristics. The standard may also be applied to Web testing, resulting in a highly generic classification of the quality dimension.

Thus, for a generic classification independent from a particular system we suggest to use the quality characteristics proposed by the ISO/IEC 9126 standard, respectively a representative subset – those which are of main importance to the user: functionality, reliability, usability, and efficiency. To obtain an even more fine-grained view, these characteristics can be further refined according to the sub-characteristics of the ISO/IEC 9216 standard. In our example, this generic classification serves as a starting point for an adapted, project specific classification.

2.2 Feature Dimension

The features, which a system contains, are usually defined in the requirements specification as functional requirements. Strictly speaking, a feature is a set of logically related functional requirements that provides a capability to the user. Testing features is one of the main concerns of conventional software testing.

Accordingly, software testing practice and theory [15, 3, 14] provide concepts and methods for decomposing a system into testable features and for deriving appropriate test cases. A common approach is to derive tests cases from use cases. Each use case represents a meaningful piece of functionality available to the user [22], corresponding approximately to a feature in our scheme.

However, unlike conventional software projects, where the focus is mainly on functions, Web projects are to some extent document-centered [19]. Content – text and figures, images, video clips, etc. – may therefore be treated as an additional feature besides those encountered in conventional software projects. Indeed, the development of content is often part of a Web project, too, in the one or the other way, e.g., from scratch or by migration. Content encompasses not only pure data but also structuring and representation issues. Hence, due to the closely intertwining of functions and content the border between these two kinds of features cannot always be clearly drawn and, henceforth, content aspects can almost never be excluded from the features of a Web project.

Apart from the two kinds of features identified – functions and content – a third one influences the quality of Web systems: infrastructure and environment. Typical examples are server configurations, network bandwidth, the geographical location of the server, partner sites (clearing services, content supplier, etc.), or workflows associated with the system. The reason to consider the infrastructure and environment is a simple but important one: In the end, quality aspects are affected by any part of the system and thus it will make no difference for users why a system fails when it fails. All that the users will do is to notice a failure, regardless of what the actual reasons are that caused this failure.

The functions of a Web application or site are often tightly coupled to the infrastructure and environment and, yet again, the distinction may not always be obvious. Nevertheless, we suggest using functions, content, and infrastructure and environment as a generic high-level distinction, as the main purpose of this dimension is to encompass all the features that may influence quality.

2.3 Phase Dimension

Testing concerns quality aspects and features in different phases of the whole lifecycle of a system. Although every project has its specific work packages and tasks that necessitate a special workflow, the overall structure usually follows the typical engineering process divided into requirements definition, design, development, integration, installation, operation and evolution, and decommissioning [25]. One possible approach would be to use these phases to categorize the phase dimension. An alternative approach would be to focus solely on testing, aligning the tests according to test phases respectively test levels, e.g., unit testing, integration testing, system testing. [15] The test levels also refer to the different points in the lifecycle at which testing takes place, although they primarily indicate the granularity of the objects to test – e.g. independent units, sub-systems, or the system as a whole. Activities like test planning, test case design, or managing test results are not explicitly considered in this approach.

The intention of the proposed scheme is to organize tests rather than testing activities, but to do so in a holistic way throughout all phases of the whole lifecycle. Therefore, we suggest combining both approaches. As a result, the scheme describes what tests take place in which phase of the lifecycle. Testing activities – test planning, test case design, etc. – are different steps in the test workflow [10] and are therefore basically out of the scope of the scheme. Quite the opposite way round, the scheme is used by the activities in the different steps of the test workflow. How this is done will be explained in section 3.

For a simplified generic overview of the phase dimension, we recommend to join the phases of the lifecycle together to at least three main lifecycle sections, namely, specification and development, testing and installation, and operation and maintenance.

3 Handling the Three-Dimensional Scheme

In this section we describe how the proposed three-dimensional cube representing the scheme for Web testing can be applied by means of two-dimensional matrices. The explanation is supported by an example presenting the application of the scheme for compiling and maintaining a set of method types pertinent for Web testing.

The question how a certain quality aspect of a certain feature can be tested in a certain phase leads straight to a clearly defined point of intersection within the presented cube (Fig. 1). To answer this question, the tests should therefore be included accordingly into the cube at the intersection points. Although it is straightforward to imagine how this works in theory, the practical realization on paper may be complicated. Hence, we use matrices to handle the three-dimensional scheme.

Three different combinations are possible whereby each is a projection of the three-dimensional cube onto a two-dimensional matrix, or, in other words, a certain view of the whole scheme: (a) the combination quality and feature, (b) the combination quality and phase, and (c) the combination feature and phase. As we focus on quality aspects, the first two combinations are of major interest. These combinations demonstrate (a) which features should be tested to assure a certain quality aspect and what tests should be applied, and (b) when or in what phases this should be done. The third combination (c) can be derived from the other two. It points out tests for each of the features in all the phases.

The matrix combining quality and features (a) can be read either from the viewpoint of quality aspects or from the viewpoint of features. Both viewpoints have to be defined, e.g. in the requirements analysis phase, so tests can be specified for the according intersection points. In this example, we use the generic categorization introduced in the previous section to give a high-level overview of Web testing in general, assigning exemplary types of test methods for Web testing to the fields of the matrix instead of specifying explicit tests.

Table 1 illustrates the corresponding quality[1]-feature matrix containing possible types of test methods. Most of these tests cover the functions as part of a system's features, which is not surprising since these tests can draw from the long tradition of software testing. Nevertheless, testing functions is still a major challenge for Web testing despite the importance of quality aspects. If the table does not list tests for a field this does not necessarily mean that no tests exist at all but rather that testing is simply less usual. This is either because there is little demand for testing this particular combination or because other quality assurance measures are preferable. The broad categories of tests mentioned in Section 1 – usability testing, security testing, performance testing, etc. – can also be found in Table 1 by grouping related aspects. Now, however, it is obvious which quality aspects and features are affected and methods and approaches for testing are clearly pointed out.

The matrix combining quality and phases (b) can be used to give an overview of which quality aspects are subject to testing during development (e.g., by unit and integration tests), when the system has been completed (e.g., by system and acceptance tests), and during operation and maintenance (e.g., by regression tests and monitoring). Again, this matrix can be read in two ways: On the one hand, it shows for every stage of the lifecycle the tests that could be applied to cover the different quality aspects, and, on the other hand, it shows for every quality aspect the tests that are appropriate in the different lifecycle stages. Yet again, ideally both viewpoints are considered contemporaneously.

Obvious in (Table 2) is the focus on tests for the completed system. These are classical tests anyway and therefore most of the tests fall into this category. In addition, all those tests that are suitable to be applied to development phases are also possible system or acceptance tests. Although it makes sense to run tests as early as possible since fixing bugs tends to become more expensive in later phases, it may be appropriate to re-run at least a part of the tests from development, especially for acceptance testing. The categorization should not be taken as a hard rule anyway. Many of the tests for the completed systems may also apply to testing sub-systems during development in earlier phases as well as they may be reused during maintenance in later phases. The column operation and maintenance lists mainly those tests that should be executed periodically as part of the system monitoring or at least every time the system is maintained, e.g. at the installation of updates and bug fixes. For further development activities that include more than a few minor changes, a whole set of new tests is usually required. However, it is important to consider the demand for tests even in the operations phase and to realize synergies over all the phases to save testing efforts and costs.

As stated above, the fields that have been grayed out, i.e. where no tests are listed, indicate those combinations where the proposed tests are less applicable, either because there is little demand for testing at an early or later phase or other quality assurance measures may be better suited. This is particularly noticeable for the quality aspects in the categories reliability, usability, and efficiency. It may be an interesting conclusion that once these quality aspects are ensured, only little change is usually encountered.

[1] The table uses the sub-characteristics of the ISO/IEC 9126 standard as classification.

Table 1. Exemplary types of test methods for the combination of quality aspects and features

		Functions	Content	Infrastructure & Environment
Functionality	Suitability	Exploratory testing GUI testing	Visual browser validation Checking for blacklisted cont. Style guide testing Accessibility testing	
	Accuracy	Boundary condition testing State transition testing GUI testing	Stylistic and lexical testing Link testing	Link testing
	Interoperability	Cross-browser compatibility Cross-platform compatibility	Visual browser validation Test printing Accessibility testing	Cross-browser compatibility Cross-platform compatibility Legacy system integration
	Compliance	Cross-browser compatibility Style guide testing	Syntax testing Style guide testing Cross-browser compatibility Accessibility testing	Cross-browser compatibility Cross-platform compatibility Legacy system integration
	Security	Common attacks Checking client for sensible data, Communication security		Network and server security
Reliability	Maturity	Endurance testing		Endurance testing
	Fault Tolerance	Forced-error testing Stress testing		Forced-error testing Low-resource testing Stress testing
	Recoverability	Forced-error testing Fail-over testing		Fail-over testing Forced-error testing Low-resource testing
Usability	Understandability	Usability studies Heuristic evaluation	Readability testing Heuristic evaluation Usability studies	
	Learnability	Usability studies Heuristic evaluation		
	Operability	Usability studies Heuristic evaluation		Heuristic evaluation
	Attractiveness		Publicity testing	
Efficiency	Time Behavior	Load testing Stress testing		Load testing Stress testing
	Resource Utilization	Endurance testing	Fast-load testing	Endurance testing Low-resource testing

Table 2. Exemplary types of test methods for the combination of quality aspects and phases

		Development	System Completed	Operation & Maintenance
Functionality	Suitability	Style guide testing Checking for blacklisted cont. Visual browser validation Accessibility testing	Exploratory testing Visual browser validation GUI testing Checking for blacklisted cont. Accessibility testing	Checking for blacklisted cont. GUI testing (regression)
	Accuracy	Stylistic and lexical testing Boundary condition testing	Boundary condition testing State transition testing GUI testing Stylistic and lexical testing Link testing	Link testing Stylistic and lexical testing GUI testing (regression)
	Interoperability	Legacy system integration Accessibility testing Visual browser validation Test printing Cross-browser testing	Visual browser validation Cross-browser and Cross-platform testing, Test printing Legacy system integration Accessibility testing	Cross-browser testing Cross-platform testing
	Compliance	Style guide testing Syntax testing Accessibility testing Cross-browser testing	Style guide testing, Syntax testing, Cross-browser and platform testing Legacy system integration Accessibility testing	Cross-browser testing Cross-platform testing
	Security	Common attacks Communication security Information security	Common attacks, Checking client for sensible data Communication security, Network and server security	Common attacks Network and server security Information security
Reliability	Maturity		Endurance testing	
	Fault Tolerance		Forced-error testing Stress testing Low resource testing	
	Recoverability		Fail-over testing Forced-error testing Low-resource testing	Fail-over testing
Usability	Understandability	Usability studies Heuristic evaluation	Usability studies Heuristic evaluation Readability testing	
	Learnability	Usability studies Heuristic evaluation	Usability studies Heuristic evaluation	
	Operability	Usability studies Heuristic evaluation	Usability studies Heuristic evaluation	
	Attractiveness		Publicity testing	Publicity testing
Efficiency	Time Behavior		Load testing Stress testing	Load testing
	Resource Utilization		Fast-load testing Endurance testing Low-resource testing	

4 Integrating the Scheme into the Test Workflow

To systematically support testing, the proposed scheme has to be integrated into the test workflow. A typical test workflow encompasses following steps: Test planning, test design, test implementation, test execution, and evaluating the results [10]. Primarily, the first two steps of this workflow will benefit from the application of the scheme. To highlight the integration of the scheme into the test workflow we demonstrate how test cases are designed for a sample Web application.

The example we selected is a simple Web-based application named *Event Calendar* serving as a demonstration application for an in-house developed Web application framework. Simplified, the Event Calendar includes the following core use cases: *View events* lists all the events for a selected month, week, or day; *View details* shows detailed information about a selected event; *Add new event* and *Edit event* allow to create or modify events incorporating the use cases *Attach document* and *Set security properties; Delete event* removes an event from the calendar; and, finally, *Logon* allows user to logon to the application.

4.1 Test Planning

One of the primary inputs for testing is the requirements specification, developed in the requirements analysis phase from the stakeholders' concerns. Requirements not only determine what should be built but – as a logical consequence – also what should be tested. In our scheme requirements constitute two of the proposed dimensions. Functional requirements are aligned along the feature dimension. Non-functional requirements are aligned along the quality dimension. Furthermore, the requirements analysis should also provide information about the dependencies between functional and non-functional requirements, or in terms of the scheme, features and quality attributes. This is exactly, what the quality-feature matrix describes: A mark or value in a field of the matrix indicates the existence or the lack of a dependency between the according combination of a quality aspect and a system feature. If the requirements specification is incomplete or missing - a problem we have encountered in several Web projects especially for non-functional requirements – the generic proposal presented above may be used to determine requirements relevant for testing and to find omissions in an incomplete requirements specification. Thus, setting up a quality-feature matrix will systematically support this digging for requirements.

Once the quality-feature matrix is available it can be used as an input for the first step in the test workflow and all the others, respectively. The matrices representing the scheme can thereby be considered as 'data structures' holding the information required for the according 'operations', i.e. activities, part of the test workflow.

Test planning is the first step of the test workflow. The purpose of test planning is to define the test strategy, to plan the required resources, and to schedule the testing efforts. The results of test planning are described in the test plan. The same information that is documented in the test plan should also be part of the two quality-feature matrix and the quality-phase matrix. In the first matrix the priorities for testing the quality and feature combinations are described, depending on the degree of dependence elicited in the requirements analysis. Techniques like ABC-analysis or pair-wise comparison should guide a systematic definition of these priorities. The second matrix illustrates how testing should be aligned along the time dimension, i.e. the amount of testing of quality aspects in the different phases. The matrices showing types of test methods introduced in the previous section may be used as a rough guidance for test planning.

Table 3 shows the according quality-feature matrix containing priorities resulting from test planning for our example Web application, the Event Calendar. This matrix focuses on security-related quality aspects, showing solely a detail view of the Event Calendar's entire quality-aspects matrix. We recommend this additional refinement by 'zooming in' as a practical presentation aid, since the entire quality-feature matrix based on the proposed generic scheme is usually too bulky to be drawn on a single sheet. The selected security-related aspects listed on the quality dimension in Table 3 have been distilled from taxonomy of common criteria for security [6]. The features making up the feature dimension originate from the aforementioned use cases of the Event Calendar example.

Table 3. shows a quality-feature matrix containing testing priorities for the Event Calendar example. The letters *A*, *B*, *C* in the matrix fields describe the priorities for testing: *A* indicates high priority, *C* low priority. Empty fields, i.e. those that have been grayed out, signify combinations that are considered not applicable or relevant

Feature / Quality aspect	**View events**	**View details**	**Add event**	**Edit event**	**Attach document**	**Set security properties**	**Delete event**
General identity	B	B	C	C	C	C	C
Message content authenticity	B	B	C	C	C	C	C
Message content origin	C	C	B	B	B	A	B
Integrity	C	C	A	A	A	A	▨
Secrecy and privacy	B	B	C	B	C	C	C
Accountability	▨	▨	B	B	B	B	B

4.2 Test Case Design

After test planning, test design is the next step in the test workflow. The main activity in this step is designing a minimal set of test cases the have a high potentiality to reveal important faults[2]. For every test case a defined goal – the verification of a quality aspect – and a defined test object – a system feature – has to be specified. Hence, by using the information provided by the quality-feature matrix the distinct goal of every test case is clearly identified and, furthermore, the design activity itself can be systematically guided to avoid omitting relevant quality aspects or features. Furthermore, deriving tests from a quality-feature matrix stemming from the requirements analysis supports the traceability of test cases.

Nevertheless, designing appropriate test cases is a creative task that requires testing and domain knowledge as well as experience. We therefore conducted brainstorming sessions in cooperation with security experts and designers to discuss potential security threats for all the relevant combinations in the quality-feature matrix. Based on the brainstorming results the actual tests were designed with the intention to reveal the existence of such security threats.

The application of the generic scheme in form of the quality-feature matrix showed that – because all relevant combinations had to be considered explicitly – many security threats were revealed that would otherwise remain undiscovered. Furthermore, this approach forces to design cases that test whether the implementation of the application features satisfy the required security aspects rather than verifying solely the correct implementation of security mechanisms. Thus, also those threats should be discovered that circumvent the implemented security mechanisms.

For the Event Calendar example we designed 26 test cases related to security aspects; a total of 80 test cases were designed for the entire Event Calendar to cover all relevant combinations of quality aspects and features. As a general rule-of-thumb we designed two test cases for every priority *A* combination of quality aspects and features and one test case for every priority *B* combination.

This example deals solely with system testing within the second phase – testing and installation – of the proposed generic scheme. However, test design for the other phases follows an analogous proceeding, albeit based on different, e.g. white-box, test techniques. For example, unit tests have been designed, too, for the security layer of the Web application framework that focuses on correctness of the implemented security mechanisms – role based access control, data encryption, etc. The developers themselves designed the tests in this phase guided by the quality-feature matrix presented in Table 3.

5 Summary and Conclusions

In this paper we presented a scheme to organize tests for quality aspects of Web applications and Web sites. The underlying idea is to distinguish between quality aspects and features, each representing a separate dimension of a three-dimensional cube. Thereby it becomes clear that the features are principally the test objects – those parts that are actually executed or analyzed in testing – while the quality aspects are

[2] Finding all bugs in a moderately complex application is usually not possible [15] and, more then ever, made difficult by the severe time and resource restrictions of many Web projects.

rather related to the test goals – they define what the tests actually try to validate and verify. The third dimension of the cube is the time dimension. This dimension specifies when testing should actually be done. How testing should be done is described by the tests assigned to the scheme. To assign tests to the scheme we used matrices to project the three-dimensional scheme onto a two-dimensional plane.

First of all, the proposed approach offers a high-level and systematic overview of Web testing in general, taking some of the recently discussed aspects of Web testing into consideration, e.g., the influence of quality aspects or the role of testing throughout the different phases and for monitoring. Web testing practice may benefit from the scheme by using it as an aid to compile and maintain a set of methods and tools. Once applied to a certain project, the presented approach supports test planning and test case design. The scheme can support test planning by specifying test goals and testing priorities in relation to quality aspects, system features, and lifecycle phases. Furthermore, we showed how the matrix combining quality aspects and system features may be integrated into the test workflow to derive quality-oriented test cases that can be traced back to related non-functional as well as functional requirements.

Applying the scheme in a real-world Web project showed that guidance explicitly considering quality aspects in addition to functions is helpful to avoid an over-reliance on functional tests. Since composing the matrix bases on the specified requirements, it also forces to systematically reconsider and review these requirements. Thus, neglected aspects and omissions in the requirements specification can be revealed. However, we also found that a tight cooperation between requirements engineering and testing is necessary to effectively benefit from the application of this scheme. Hence, one open issue to investigate is the adaptation of this approach for requirements engineering in particular and throughout the Web development process in general.

The usage of this scheme is not restricted to Web testing. However, we see Web projects as the main field of application as these projects typically have to satisfy a wide range of quality aspects with a limited set of resources demanding a lightweight and adaptable approach.

References

1. Atzeni, P., Merialdo, P., Sindoni, G.: Web Site Evaluation: Methodology and Case Study. In: Proc. of the Int. Workshop on Data Semantics in Web Information Systems. 20th Int. Conference on Conceptual Modeling, Yokohama, Japan (November 2001)
2. Bazzana, G.: Ensuring the quality of Web Sites and E-commerce applications. In: Wieczorek, M., Meyerhoff, D. (eds.): Software Quality: State of the Art in Management, Testing, and Tools. Springer-Verlag, Heidelberg (2001)
3. Beizer, B.: Software Testing Techniques. 2.ed., Van Nostrand Reinhold (1990)
4. Boehm, B., In, H.: Identifying Quality-Requirement Conflicts. In: IEEE Software, Vol. 13, 2 (March 1996) 25-35
5. Boehm, B., Brown, J.R., Lipow, M.: Quantitative Evaluation Of Software Quality. 2nd Int. Conference on Software Engineering, San Francisco (1976)

6. Common Criteria: Common Criteria for Information Technology Security Evaluation, (http://www.commoncriteria.org/), August 1999
7. Dustin, E., Rashka, J., McDiarmid, D.: Quality Web Systems: Performance, Security, and Usability. Addison-Wesley (2001)
8. Ginige, A., Murugesan, S.: Web Engineering: An Introduction. IEEE Mulitmedia, vol.8, no.1, IEEE (Jan.-March 2001) 14-18
9. ISO/IEC Std. 9126-1:2001: Software engineering – Product quality – Part 1: Quality model. International Organization of Standardization (2001)
10. Jacobson, Booch, Rumbaugh: The Unified Software Development Process. Addison-Wesley (1999)
11. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. Wiley&Sons, New York (1998)
12. Kung, D.C., Liu, C.-H., Hsia, P.: An Object-Oriented Web Test Model for Testing Web Applications. In: Proceedings of APAQS'00, Hongkong, China, IEEE (October 2000)
13. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in Software Quality, Vol. I-III; TR-77-369, Rome Air Development Center (1977)
14. McGregor, J.D., Sykes, D.A.: A Practical Guide to Testing Object-Oriented Software. Addison-Wesley (2001)
15. Myers, G.J.: The Art of Software Testing. Wiley (1979)
16. Olsina, L.: Web-site Quality Evaluation Method: a Case Study on Museums. ISCE 99, 2nd Workshop on Software Engineering over the Internet
17. Olsina, L., Godoy, D., Lafuente, G.J., Rossi, G.: Quality Characteristics and Attributes for Academic Web Sites; WWW8 Web Engnieering '99 Workshop, Toronto, Canada, May 1999
18. Olsina, L., Godoy, D., Lafuente, G.J., Rossi, G.: Specifying Quality Characteristics and Attributes for Websites. In: Murugesan, S., Deshpande, Y. (eds.): WebEngineering 2000, LNCS 2016. Springer-Verlag, Heidelberg (2001) 166-278
19. Powell. T.A.: Web Site Engnieering: Beyond Web Page Design. Prentice-Hall, Upper Saddle River NJ (1998)
20. Pressman, R.S.: What a Tangled Web We Weave. IEEE Software, vol.17, no.1 (Jan-Feb 2000) 18-21
21. Rudolf, A., Pirker, R.: E-Business Testing: User Perceptions and Performance Issues. In: Proceedings of APAQS'00, Hongkong, China, IEEE (October 2000)
22. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley (1998)
23. Schach, S.: Testing: Principles and Practice. In: Tucker, A.B.: The Computer Science and Engineering Handbook. CRC Press, Boca Raton FL (1996)
24. Sommerville, I: Software Engineering. 6.ed., Addison-Wesley (2001)
25. Splaine, S., Jaskiel, S.P.: The Web Testing Handbook. STQE Publishing (2001)
26. Subraya, B.M., Subrahmanya, S.V.: Object driven Performance Testing of Web Applications. In: Proceedings of APAQS'00, Hongkong, China, IEEE (October 2000)