7 Testing Web Applications

Christoph Steindl, Rudolf Ramler, Josef Altmann

Web applications have developed into an essential communication platform for many companies. Web applications are crucial for commerce, information exchange, and a host of social activities. For this reason Web applications have to offer high-performance, reliable, and easy-to-use services round the clock. Offering excellent Web applications for existing and future users represents a major challenge for quality assurance. Testing is one of the most important quality assurance measures. Traditional test methods and techniques concentrate largely on testing functional requirements. Unfortunately, they do not focus enough on the broad range of quality requirements, which are important for Web application users, such as performance, usability, reliability, and security. Furthermore, a major challenge of testing Web applications is the dominance of change. User requirements and expectations, platforms and configurations, business models, development and testing budgets are subject to frequent changes throughout the lifecycle of Web applications. It is, therefore, necessary to develop an effective scheme for testing that covers the broad range of quality characteristics of Web applications and handles the dominance of change, helping to implement and better understand a systematic, complete, and risk-aware testing approach. Such a test scheme forms the basis for building an exemplary method and tool box. Practical experience has shown that methodical and systematic testing founded on such a scheme is feasible and useful during the development and evolution of Web applications.

7.1 Introduction

Web applications pose new challenges to quality assurance and testing. Web applications consist of diverse software components possibly supplied by different manufacturers. The quality of a Web application is essentially determined by the quality of each software component involved and the quality of their interrelations. Testing is one of the most important instruments in the development of Web applications to achieve high-quality products that meet users' expectations.

Methodical and systematic testing of Web applications is an important measure, which should be given special emphasis within quality assurance. It is a measure aimed at finding errors and shortcomings in the software under test, while observing economic, temporal, and technical constraints. Many methods and techniques to test software systems are currently available. However, they cannot be directly applied to Web applications, which means that they have to be thought over and perhaps adapted and enhanced.

Testing Web applications goes beyond the testing of traditional software systems. Though similar requirements apply to the technical correctness of an application, the use of a Web application by heterogeneous user groups on a large number of platforms leads to special testing requirements. It is often hard to predict the future number of users for a Web application. Response times are among the decisive success factors in the Internet, and have to be tested early, despite the fact that the production-grade hardware is generally available only much later. Other important factors for the success of a Web application, e.g., usability, availability, browser compatibility, security, actuality, and efficiency, also have to be taken into account in early tests.

This chapter gives an overview of solutions, methods, and tools for Web application testing. The experiences from several research projects and a large number of industrial projects form the basis for the development of a structured test scheme for Web applications. Starting with software testing basics in section 7.2, section 7.3 will use practical examples to discuss the specifics in Web application testing. Section 7.4 describes the impact of conventional and agile test methods on Web applications. Section 7.5 describes a generic test scheme that extends the focus of testing to a broad range of quality characteristics. Section 7.6 uses this test scheme to give an overview of methods and techniques for Web application testing. Section 7.7 discusses automation and the use of tools in Web application testing. Finally, section 7.8 closes this chapter with an outlook.

7.2 Fundamentals

7.2.1 Terminology

Testing is an activity conducted to evaluate the quality of a product and to improve it by identifying defects and problems. If we run a program with the intent to find errors, then we talk about *testing* (Myers 1979). Figure 7-1 shows that testing is part of analytical quality assurance measures (Bourque and Dupuis 2005). By discovering existing errors, the quality state of the program under test is determined, creating a basis for quality improvement, most simply by removing the errors found.

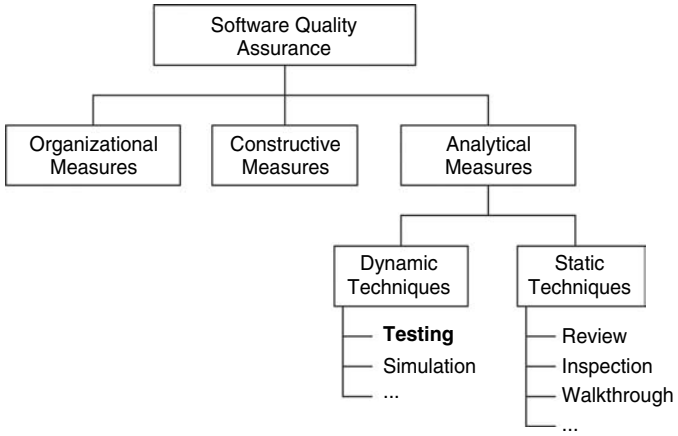

Figure 7-1 Structuring software quality assurance.

relevant knowledge should additionally be discovered and reused, e.g., by so-called recommender systems (Kobsa et al. 2001). At the same time, an increasing number of approaches in the field of *usage analysis* for Web applications have been suggested, resulting from the analysis of accesses and user behavior, aimed at largely automating the improvement measures (so-called "adaptive systems" Oppermann and Specht 1999). In this respect, we can basically distinguish whether such an adaptation is static, i.e., before a user accesses a Web application, or dynamic, i.e., at the time when it is accessed (Kappel et al. 2003). In any event, the degree of automation should be adapted to the type of adaptation to avoid annoying users (Kobsa et al. 2001). (Mobasher et al. 2000 and Niu et al. 2002) suggest automatic personalization at runtime on the basis of user interaction clustering. Open issues in this context include the embedding of artificial intelligence techniques to develop adaptive systems and the performance of such systems.

We say that an *error* is present if the actual result from a test run does not comply with the expected result. The expected result is specified, for example, in the requirements definition. This means that each deviation from the requirements definition is an error; more generally speaking, an error is "the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition" (IEEE standard 610.12-1990).

This definition implies that the requirements definition used as a basis for testing is complete and available before implementation and test. A common phenomenon in the development of Web applications is that the requirements are often incomplete, fuzzy, and subject to frequent changes. Typically, there is an initial vision of the basic functionality. This vision is implemented for the initial release. As a result, the initial development lifecycle is followed by smaller cycles of functionality additions. Agile approaches (such as Extreme Programming, see Chapter 10 and (Highsmith 2002) for a general overview) focus on this iterative and evolutionary nature of the development lifecycle without an extensive written requirements definition. Consequently the goals, concerns, and expectations of the stakeholders have to form the basis for testing. This means that, for example, each deviation from the value typically expected by users is also considered an error.

Now, different stakeholders generally have different expectations, and some of these expectations may even be competing and fuzzy. For this reason, stakeholder expectations won't be a useful guideline to decide whether a result is erroneous unless agreement on a set of expectations has been reached and made available in testable form (see Chapter 2). To support the tester in gaining insight into the users' world and to better understand users' expectations, the tester should be involved as early as possible in the identification and definition of requirements.

When talking about a *test* in the further course of this chapter, we mean a set of test cases for a specific object under test (i.e., a Web application, components of a Web application, or a system that runs a Web application). A single *test case* describes a set of inputs, execution conditions, and expected results, which are used to test a specific aspect of the object under test (IEEE standard 610.12-1990).

7.2.2 Quality Characteristics

A user does not only expect an application to behave in a certain way; he or she also expects that certain functions are available 24 hours per day and 7 days a week (24x7). Moreover, users expect the application to be easy to use, reliable, fast, compatible with other systems and future versions, and the like. In addition to the behavior it is, therefore, important to test the application as to whether or not it meets its quality requirements, i.e., the kinds of quality characteristics expected by users.

Chapter 2 described the different quality characteristics in the context of Web applications. A general taxonomy for quality characteristics of software products is specified in the ISO/IEC 9126-1 standard. This standard mentions six principal categories of characteristics – functionality, reliability, usability, efficiency, maintainability, and portability – and breaks them down further into sub-characteristics.

Quality requirements play an essential role when testing Web applications. Though they are generally similar to quality requirements for traditional software systems, they often reach beyond them in both their breadth and depth (Offutt 2002). Due to the great significance of distinct quality characteristics and the differences as to how they can be tested, many methods for

Web application testing concentrate on one or a few specific quality characteristics. However, all quality characteristics are important for the overall quality of a Web application. Testing has to ensure that they are successfully implemented.

7.2.3 Test Objectives

Testing won't lead to quality improvement unless errors are detected and removed. The main test objective is to find errors, rather than to show their absence. Software tests are unsuitable to prove the absence of errors. If a test doesn't find errors, then this does not mean that the tested application doesn't contain any. They may simply not have been detected yet.

The large number of quality characteristics to be considered, and all potential input values and input combinations, including all potential side conditions and processes, make it impossible to achieve complete test coverage. Even broad test coverage is typically impossible within the often extremely short development cycles. The inevitable consequences are flaws in tested functions and a higher risk of errors persisting undetected. These are the reasons why testing tends towards a risk-based approach. Those parts of an application where errors go undetected, and where these errors would have the most critical consequences, should be tested first and with the greatest effort. Exploring the sources of risk may point to defects more directly than basing tests mainly on requirements (Bach 1999). As a consequence, a further important test objective is to bring that risk to light, not simply to demonstrate conformance to stated requirements.

A test run is successful if errors are detected, respectively additional information about problems and the status of the application is acquired. Unsuccessful tests, i.e., tests that do not find errors, are "a waste of time" (Kaner et al. 1999). This is particularly true in Web application development, where testing is necessarily limited to a minimum due to restricted resources and the extreme time pressure under which Web applications are developed. This situation also requires that serious errors should be discovered as early as possible to avoid unnecessary investments as the cost of finding and removing errors increases dramatically with each development phase (Kaner et al. 1999). Errors that happened in early development phases are hard to localize in later phases, and their removal normally causes extensive changes and the need to deal with consequential errors. Therefore we have to start testing as early as possible at the beginning of a project.

In addition, short time-to-market cycles lead to situations where "time has to be made up for" in the test phase to compensate for delays incurred in the course of the project. Testing effectiveness and the efficiency of tests are extremely important. In summary, we can say that testing in general, and for Web projects in particular, has to detect as many errors as possible, ideally as many serious errors as possible, at the lowest cost possible, within as short a period of time as possible, and as early as possible.

7.2.4 Test Levels

According to the distinct development phases in which we can produce testable results, we identify *test levels* to facilitate testing of these results.

- *Unit tests*: test the smallest testable units (classes, Web pages, etc.), independently of one another. Unit testing is done by the developer during implementation.

The *pre-processing phase* typically *cleans out* the information in log file entries, e.g., by eliminating search server accesses or accesses to embedded images. The next step *identifies sessions and users*, where heuristics have sometimes to be used (see "path completion" in (Cooley et al. 1999)) due to problems in identifying users from IP addresses, or gaps in the log transformation. This already implies the acquaintance with the Web application itself. For example, page types have to be defined to distinguish content pages from pure navigation pages such as menus.

The *pattern discovery phase* identifies related pages visited by users within the same session. To improve a Web application, for example, these pages can have additional links between them. Moreover, discovering the Web pages visited consecutively is useful in improving the performance by *pre-fetching* relevant pages (Barish and Obraczka 2000). Another result from pattern discovery can be a classification of users into interest groups by using user profiles.

And finally, the *pattern analysis phase* consists of a filtering and interpretation process of the patterns found depending on the application. The analysis goal can thereby be rather complex especially for e-commerce applications, which (Kohavi 2001) addresses as the killer-domain for web usage mining. Collected usage data can only be analyzed together with additional information about the content and the structure of the Web application, and, moreover, general background information about the problem and its domain. Challenging analysis goals, therefore, demand for so-called clickstream data warehousing (Hacker 2003, Grossmann et al. 2004). These data warehouses aim, firstly, at calculating user independent statistical indicators, e.g. traffic intensity within a certain part of the Web application, or conversion rates, i.e. the number of visitors who buy something from the Web site, expressed as a percentage of total visitors. Secondly, user or user group dependent optimization potentials can be assessed, e.g., streamlining the navigation paths. And thirdly, they can also provide the basis for dynamic customization or personalization of the content, which can yield immediate economic effects. In this context the term *collaborative filtering* is used for methods that support users across intelligent Web applications, optimized to their individual navigation behavior. The options offered to a user are limited based on information gathered on other users with presumably similar behavior (collaborative aspect), or prioritized (filtering aspect). A heuristic technique typically applied in collaborative filtering (and in recommender systems based upon this technique) is based on the assumption that similar users have similar preferences. For example, if two users of an interest group book a specific tourist product on a tourist Web application, then this product could be recommended to a third user from the same interest group (Ricci 2002).

8.6 Outlook

In this chapter we have discussed several tasks related to the operation and maintenance phase of a Web application. In the following we will focus on two areas, which will have a considerable impact on further research work.

Content syndication in the sense of knowledge management based on technologies of the semantic Web increasingly gains significance (see also Chapter 14). This is not least an implication of the media convergence, i.e. the breakthrough of mobile devices demanding valuable content. It is the general objective to prepare the wealth of existing information not only in a way humans can understand, but also in a way machines can process automatically (Berners-Lee et al. 2001). Not only should existing knowledge be appropriately linked, but new

- *Visits*: A series of one or more page impressions (within one Web application), served to one user, which ends when there is a gap of 30 minutes or more between successive page impressions for that user.

Other interesting statistical indicators include the *average time a user spends* on a Web site, and basic advertising measures such as *ad impression*, which is the count of a delivered basic advertising unit from an ad distribution point, or *ad click*, which is when a visitor interacts with an advertisement, or the *ad click rate*, which is the ad click to ad impression ratio.

An estimate with regard to the popularity of a Web application can be deduced from the number of users who *bookmarked* a page. Every time a client bookmarks a Web page of your Web application using a common browser, e.g. Microsoft Internet Explorer, the favicon.ico file is accessed. This file is typically located in the Web server's root directory and contains a 16×16 pixel graphic, which is added to the bookmarked Web page on the client's browser to be viewed on request of this Web page.[8]

Many Web sites offer advertising space for sale. Advertisers pay rates based on the number of visitors and page impressions, similar to the way other media sell advertising. Potential advertisers use usage statistics like the ones mentioned above to have a better understanding of what these numbers mean. Many official audit bureaus have established as members of the *International Federation of Audit Bureaux of Circulations* (`http://www.ifabc.org`) to ensure that there are standardized measurements of visits and page impressions. For example, *ABC Electronic* (`http://www.abce.org.uk`) is a non-profit organization acting as a third-party auditing organization in the United Kingdom. Its publication audits are based on rules governing how audits are conducted and how publishers report their circulation figures. These audit bureaus typically charge a membership fee and collect access data based on page tagging based methods, i.e., the members have to include, for example, a pixel or some script code within their Web pages (cf. section 8.5.1). This effort pays off, if these types of accredited evaluations yield positive marketing effects to increase credibility.

8.5.3 User Behavior Analysis

The e-commerce environment is interested in more complex statistics, beyond visit and page impression counts. These include product classes sold, various conversion rates, number of visitors who turned into buyers, profit registered per customer, and other information.

The process of analyzing user behavior is know as *Web usage mining*. This method uses data-mining techniques to identify patterns in the usage of Web applications aimed at improving a Web application (Srivastava et al. 2000). Results gained from Web usage mining can reach from simple information, e.g., which Web page is the most frequently visited in a Web application, to complex information such as recording the path across a Web application, tracking a user until they purchase a product, or forming user groups for the purpose of collaborative filtering (Herlocker et al. 2000).

Web usage mining can build on data acquisition techniques that mostly use log files. According to (Srivastava et al. 2000), Web usage mining consists of three phases, namely pre-processing, pattern discovery, and pattern analysis.

8 `http://msdn.microsoft.com/library/default.asp?url=/workshop/Author/dhtml/howto/ShortcutIcon.asp`.

- *Integration tests*: evaluate the interaction between distinct and separately tested units once they have been integrated. Integration tests are performed by a tester, a developer, or both jointly.
- *System tests*: test the complete, integrated system. System tests are typically performed by a specialized test team.
- *Acceptance tests*: evaluate the system in cooperation with or under the auspice of the client in an environment that comes closest to the production environment. Acceptance tests use real conditions and real data.
- *Beta tests*: let friendly users work with early versions of a product with the goal to provide early feedback. Beta tests are informal tests (without test plans and test cases) which rely on the number and creativity of potential users.

As development progresses, one proceeds from a verification against the technical specification (if available) – as in unit tests, integration tests and system tests – to a validation against user expectations – as in acceptance tests and beta tests.

An inherent risk when performing the test levels sequentially according to the project's phases is that errors due to misunderstood user expectations may be found only at a late stage, which makes their removal very costly. To minimize this risk, testing has to be an integrated part of the product construction which should encompass the whole development process. Hence, quality-assurance measures like reviews or prototyping are used even before running unit tests. A strongly iterative and evolutionary development process reduces this risk since smaller system parts are frequently tested on all test levels (including those with validation against user expectations), so that errors can be found before they can have an impact on other parts of the system. This means that the sequence of test levels described above does not always dictate the temporal sequence for Web project testing but may be performed several times, e.g. once for each incrementation of functionality.

7.2.5 Role of the Tester

The intention to find as many errors as possible requires testers to have a "destructive" attitude towards testing. In contrast, such an attitude is normally difficult for a developer to have towards his or her own piece of software, the more so as he or she normally doesn't have sufficient distance to his or her own work after the "constructive" development and problem solving activity. The same perspective often makes developers inclined to the same faults and misunderstandings during testing that have led to errors during the implementation in the first place. For this reason, (Myers 1979) suggests that developers shouldn't test their own products.

In Web projects, we have an increased focus on unit tests which are naturally written by the developers. While this is a violation of Myers' suggestion, additional tests are typically performed by someone different from the original developer (e.g. by functional testers recruited from the client's business departments).

Since quality is always a team issue, a strict separation of testing and development is not advisable and has an inherent risk to hinder the close cooperation between developers and testers. After all, the objective pursued to detect errors is that errors will be removed by the developers. To this end, a clearly regulated, positive communication basis and mutual understanding are prerequisites. This means for the tester: "The best tester isn't the one who finds the most bugs

or who embarrasses the most programmers. The best tester is the one who gets the most bugs fixed." (Kaner et al. 1999).

Since Web project teams are normally multidisciplinary, and the team cooperation is usually of short duration, it can be difficult for team members to establish the necessary trust for close collaboration between developers and testers.

7.3 Test Specifics in Web Engineering

The basics explained in the previous section apply both to conventional software testing and Web application testing. What makes Web application testing different from conventional software testing? The following points outline the most important specifics and challenges in Web application testing based on the application's characteristics (see section 1.3).

- Errors in the "content" can often be found only by costly manual or organizational measures, e.g., by proofreading. Simple forms of automated checks (e.g., by a spell checker) are a valuable aid but are restricted to a limited range of potential defects. Meta-information about the content's structuring and semantics or a reference system that supplies comparative values are often a prerequisite to be able to perform in-depth tests. If these prerequisites are not available, other approaches have to be found. For example, if frequently changing data about the snow situation in a tourist information system cannot be tested by accurate meta-information or comparative values, then the validity of the data can be heuristically restricted to two days to ensure the data's actuality.
- When testing the hypertext structure, we have to ensure that the pages are linked correctly, e.g., each page should be accessible via a link and, in turn, it should have a link back to the hypertext structure. In addition, all links have to point to existing pages, i.e., they mustn't be broken. *Broken links* represent frequent errors when statically pre-defined links become invalid, for example, when an external Web page is referenced, which has been removed or changed its structure. Another source of errors is the navigation via Web browser functions, e.g., "Back in History", in combination with the states in which a Web application can be. A typical example: If a user places an article in the virtual shopping cart while shopping online, then this article will remain in the shopping cart even if the user goes one step back in the browser history, displaying the previous page without that article.
- The soft, subjective requirements on the presentation level of Web applications, e.g., "aesthetics", are difficult to specify. However, this is an essential prerequisite for the tester to be able to clearly and objectively distinguish acceptable (and desired) behavior from faulty behavior. Moreover, only a few conventional methods and techniques for software testing are suitable for presentation testing. To test a presentation, methods from other disciplines, e.g., print publishing, and organizational measures have to be used, similarly to content quality assurance.
- The large number of potential devices and their different performance characteristics (*multi-platform delivery*) represent another challenge. Even if a tester had all potential devices at disposal, he or she would have to run test cases for each device. Though simulators for devices can be helpful since the tester doesn't have to physically provide for the devices, they are often faulty themselves, or they are unable to exactly map a device's properties, or they become available only after the introduction of a device.

For example, you could identify menu pages or aborted actions as such to prevent them from being included in the statistics. The same technique can be used for arbitrary user behavior analyses, even across several (collaborating) servers. Dependent on their use, these tags are also known as *Web bugs* (`http://www.eff.org/Privacy/Marketing/web_bug.html`) or Web beacons.

Monitoring Network Traffic

In common networking industry parlance, a *packet sniffer* is a tool that monitors and analyzes network traffic to detect bottlenecks and problems. Packet sniffers permanently monitor and log the data transfer, enabling real-time analyses, so that immediate actions can be taken, e.g., when certain components of a Web application are overloaded, or when a hardware component is defective. On the other hand, they supply additional information that log files don't normally capture, e.g., when a user aborted a transmission. Packet sniffers have a few drawbacks, e.g., data loss when the packet sniffer goes down, or encrypted data packets that cannot be analyzed. Moreover, they require access to the network layer, and since they can also be used illegitimately, they can cause security risks, such as password eavesdropping.

Individual Application Extensions

The drawbacks of the techniques mentioned above mean in many cases that individual application extensions are required (Kohavi 2001). For example, extensions could allow us to use detailed logs on user registration activities, store data input in Web forms, or calculate sales generated with registered users. Most of these additional functionalities are rather costly, which means that we should consider them in early phases of the development project. As an alternative to extending applications, the e-commerce industry uses extensive analytical tools, for instance, to gather business transactions. Unfortunately, they often fail because these tools are very expensive and typically too complex to be easily integrated in Web applications.

8.5.2 Statistical Indicators

The success of a (commercial) Web application is typically measured by the number of visitors or accesses to the Web application. These measurements normally use the following statistical indicators:[7]

- *Hits*: Each element of a requested page, including images, text, interactive items, is recorded as a "hit" in the site's Web server log file. Since a Web page can contain many such elements depending on the design, this statistical indicator is not particularly meaningful.
- *Page impressions*: A single file or combination of files, sent to a valid user as a result of that user's request being received by the server, i.e. counts pages loaded completely, for example, a Web page composed of several frames is counted as *one* page impression.

7 Definitions are taken from ABC Electronic (`http://www.abce.org.uk/cgi-bin/gen5?runprog=abce/abce&noc=y`), an audit bureau for standardized measurement of Web access metrics (see also discussion at the end of the current section).

or the *Extended Logfile Format* (http://www.w3.org/TR/WD-logfile.html). Log files include the following information:

- The user's *IP address* or *host name*, which is used as a basis for general user identification.
- The *access time* (date, time, time zone).
- The *client's request*, e.g. an HTML or an image file.
- The *referrer*, which is the page the visitor viewed before making this request. The referrer URL is the basis for evaluating the navigation path.
- The *user agent*, which describes the user's software equipment, including browser and operating system types and versions.

Problems arise particularly with regard to identifying a user by the IP address of the Web client. Internet providers often assign IP addresses dynamically to Web clients, which means that a single user is represented by means of different identifications (NAT, network address translation). Vice versa, many users can act under the same identification in public Web clients. This means that it is not easy or reliable to aggregate data into individual user profiles over lengthy periods of time. Solutions to this problem use cookies, or explicit user login.

To recognize re-visits, Web server log files can store so-called *cookie* information (http://www.w3.org/Protocols/rfc2109/rfc2109) about the client which requested the page (see also the discussion of proxies in section 4.5.2). A major benefit with respect to usage analysis is that the cookie string can be used to identify sessions. If there are no cookies available, the only way to identify a session at this level is to sort out the Web log file entries by *IP-address, client browser version,* and *date and time*. Using cookies, one can at least make sure that the same client-server/browser combination is traced.

Many commercial analysis tools build on log file analysis. However, this evaluation method has a few weaknesses, in addition to the user identification problem mentioned above. Some of these weaknesses can be attributed to the original intention to use log files as debugging tools for Web server troubleshooting. Log files of heavily frequented Web sites can be *extremely large*, making both a manual and a tool-supported evaluation difficult or even impossible. For example, access-intensive Web servers produce log files with several gigabytes on a single day, where search engine robot entries usually account for a large part. For this reason, many analytical tools run evaluations only weekly or monthly. These *time-delayed analyses* are insufficient for many Web applications. Mature commercial products, e.g., Webtrends (http://www.webtrends.com), do shorter evaluations based on log files, while *on-demand Web analyses* build on other technologies.

Page Tagging Based Methods

On-demand analytical tools (also termed *live reporting tools*) such as Hitbox offered by WebSideStory (http://www.websidestory.com) are based on page tagging methods. The page-tagging methodology requires a graphic or script tag to be inserted on each page of a Web application that is intended to be tracked. Typically it is managed by a third-party server, which keeps track of how many times each tag is sent or script is executed, thus allowing real-time analyses. A common method is the use of a so-called *clearGIF*, a transparent image on a Web page, which is normally invisible because it is typically only 1-by-1 pixel in size and set in the background color. Page tagging based analysis can be more targeted than a classic log file analysis.

- Due to the global availability and usage of Web applications, there are many challenges with regard to multilinguality and usability (see Chapter 11) in Web application testing. The major challenge is to recognize cultural interdependencies and consider them adequately in the test. For example, reading orders in different cultures (e.g., Arabic, Chinese) imply specific lateral navigation aids in the browser window. Another difficulty stems from different lengths of text messages in different languages which may result in layout difficulties.
- The common "juvenility" and "multidisciplinarity" of teams are often tied to poor acceptance of methodologies and poor readiness to do testing. Often knowledge about methods, technologies, and tools has to be acquired in the course of a project. Different points of view with regard to testing have to be consolidated. Only a team of sensitive and experienced members will come to a good decision about the amount of testing – too much testing can be just as counterproductive as too little. Testers are often tempted to test everything completely, especially at the beginning.
- Web applications consist of a number of different software components (e.g., Web servers, databases, middleware) and integrated systems (e.g., ERP systems, content management systems), which are frequently supplied by different vendors, and implemented with different technologies. These components form the technical infrastructure of the Web application. The quality of a Web application is essentially determined by the quality of all the single software components and the quality of the interfaces between them. This means that, in addition to the components developed in a project, we will have to test software components provided by third parties, and the integration and configuration of these components. Many errors in Web applications result from the "immaturity" of single software components, "incompatibility" between software components, or faulty configuration of correct software components. (Sneed 2004) reports on an industrial project where the test strategy was to test the compatibility of the Web application with the technical environment in addition to testing the functionality of the Web application.
- The "immaturity" of many test methods and tools represents additional challenges for the tester. If a Web application is implemented with a new technology, then there are often no suitable test methods and tools yet. Or if initial test tools become available, most of them are immature, faulty, and difficult to use.
- The "dominance of change" makes Web application testing more complex than conventional software testing. User requirements and expectations, platforms, operating systems, Internet technologies and configurations, business models and customer expectations, development and testing budgets are subject to frequent changes throughout the lifecycle of a Web application. Adapting to new or changed requirements is difficult because existing functionality must be retested whenever a change is made. This means that one single piece of functionality has to be tested many times, speaking heavily in favor of automated and repeatable tests. This places particular emphasis on regression tests, which verify that everything that has worked still works after a change. Upgrades and migrations of Web applications caused by ever-changing platforms, operating systems or hardware should first run and prove successful in the test environment to ensure that there will be no unexpected problems in the production environment. A second attempt and an ordered relapse should be prepared and included in the migration plan – and all of this in the small time window that remains for system maintenance, in addition to 24x7 operation ("availability").

7.4 Test Approaches

Agile approaches (such as Extreme Programming, see Chapter 10 and (Highsmith 2002) for a general overview) have increasingly been used in Web projects. While agile approaches focus on collaboration, conventional approaches focus on planning and project management. Depending on the characteristics of a Web project, it may be necessary to perform test activities from agile and conventional approaches during the course of the project. (Boehm and Turner 2003) describe at length how to find the right balance between agility and discipline on projects. This section will not introduce one specific approach for Web application testing. Instead, we will explain the characteristics of conventional and agile testing approaches, and show how they differ.

7.4.1 Conventional Approaches

From the perspective of a conventional approach, testing activities in a project include planning, preparing, performing, and reporting:

- *Planning*: The planning step defines the quality goals, the general testing strategy, the test plans for all test levels, the metrics and measuring methods, and the test environment.
- *Preparing*: This step involves selecting the testing techniques and tools and specifying the test cases (including the test data).
- *Performing*: This step prepares the test infrastructure, runs the test cases, and then documents and evaluates the results.
- *Reporting*: This final step summarizes the test results and produces the test reports.

On the one hand, conventional approaches define work results (e.g., quality plan, test strategy, test plans, test cases, test measurements, test environment, test reports) and roles (e.g., test manager, test consultant, test specialist, tool specialist) as well as detailed steps to create the work results (e.g., analyze available test data or prepare/supply test data). Agile approaches, on the other hand, define the quality goal and then rely on the team to self-organize to create software that meets (or exceeds) the quality goal.

Due to the short time-to-market cycles under which Web applications are developed, it is typical to select only the most important work results, to pool roles, and to remove unnecessary work steps. It is also often the case that "time has to be made up for" in the test phase to compensate for delays incurred in the course of the project. Therefore, test activities should be started as early as possible to shorten the critical path – the sequence of activities determining the project duration (IEEE standard 1490–1998) – to delivery. For example, planning and design activities can be completed before development begins, and work results can be verified statically as soon as they become available. Figure 7-2 shows that this helps shorten the time to delivery, which complies nicely with the short development cycles of Web applications.

7.4.2 Agile Approaches

Agile approaches assume that a team will find solutions to problems jointly and autonomously (reliance on self-organization). This also applies to testing. Therefore, testing is not a matter of

8.4.2 Content Syndication

Content syndication means the supply of content (e.g. currency data) or of an application (e.g., a route planner) by a *syndicator* for reuse in a Web application by a *subscriber*. A content syndicator and a content subscriber normally sign an agreement governing issues like copyright delivery terms and conditions, and billing, which is often based on flat-fee licensing or page-impressions (see section 8.5.2).

In the course of developing and maintaining a Web application, we should consider that the content generally has to be customized to the subscriber's structure and presentation requirements (Stonebraker and Hellerstein 2001). In addition, the supplied content may have to be personalized to target groups. For a standardized exchange of information, we could use, for example, the XML-based *Information and Content Exchange* (*ICE*) protocol (`http://www.icestandard.org`). ICE facilitates the controlled exchange and management of electronic assets between partners and affiliates across the Web.

8.5 Usage Analysis

(Web) usage analysis (also termed (Web) access analysis) is understood as the process of assessing and analyzing the use of a Web application. Usage analysis serves to measure a Web application's success by evaluating various indicators. It is further used to yield information about the users and their behavior, representing a basis for improvements in the Web application, e.g., with regards to navigation and performance, and even adaptation to specific user groups.

8.5.1 Usage Analysis Techniques

To acquire data for the purpose of conducting a usage analysis, we can basically distinguish between *explicit (or offline) methods*, which are based on direct user input, and *implicit (or online) methods*, which collect data (automatically) without involving users. Examples of explicit methods are regular *online surveys* based on Web forms offered by many agencies, and *e-mail evaluation* where users can report problems that occurred with a Web application. These are effective methods for individual survey and evaluation, but their implementation is typically costly, and their main shortcoming lies in the fact that they require users to actively participate. Therefore, the common techniques used for usage analysis are diverse implicit methods, which are discussed in the following.

Web Server Log File

Probably the most important implicit method is based on Web server log files, which record all accesses to a Web site and its units (Web pages, frames, etc.). When we say *log file analysis*, we speak of the evaluation of these log files, for instance, to determine indicators or the navigation behavior of a specific user.

Log files can have different formats, e.g., the *Common Logfile Format* (*CLF*) as a minimum standard (for a detailed description see also `http://www.w3.org/Daemon/User/Config/Logging`),

8.4.1 Content Update Rate and Demand on Currency

While the structure and presentation of a Web application are subject to certain change requests, they are generally kept stable over a lengthy period of time. In contrast, the contents of most Web applications are subject to permanent changes. The quality aspects particularly relevant in the operation and maintenance phase are the content's *volume*, constantly requiring the creation of new content, and its *currency*,[6] which is closely related to the content's *consistency* and *reliability* (Wang et al. 1995).

Ensuring that content is current means that we have to observe various update cycles, depending on the Web application type. By nature, these update cycles are much shorter for a news site, where editing is necessary as soon as information changes ("on demand"), or hourly. For other sites, such as a company presentation, a daily, weekly, monthly, or even quarterly update rate may be sufficient. However, it is not sufficient to merely look at the type of Web application, especially for e-commerce applications. We have to consider currency requirements on the level of single content units (text, images, etc.) and try to meet them (see also Chapter 1).

Poor content currency can cause users to lose confidence in the site. For example, a travel-industry Web application includes information units that have different change frequency requirements and different currency needs. Availability information may be subject to permanent change, while pricing information changes seasonally. For both, currency demand is high because this information forms the basis for business transactions when users make online bookings. Incorrect prices may lead to cancellations. Outdated availabilities may lead to situations where free capacities are not seen as such and, therefore, not sold; at the other end of the spectrum, they may lead to overbooking when capacities listed don't actually exist. In contrast, the description of a hotel or vacation home is not likely to change often, and would probably not have negative consequences for users or site owners if they weren't current.

If we use a content management component, then the time when we manage the content and the time when the new or modified content appears on the Web pages may differ, which has a negative impact on how users perceive content currency. This depends on the strategy used to generate Web pages. Generating a Web page on the fly i.e. dynamically at the time of a user's request ensures maximum currency, but can lead to poor performance at runtime. Generating a Web page periodically, e.g., once daily, and storing it, e.g., in the file system, also referred to as Web page materialization, leads to shorter response times and higher stability because the updated Web pages are already available when they are requested by users. Periodic generation, however, is lacking in data consistency and is meaningful only for content with low currency demands. Generating a page at the time when the content is managed combines the advantages of both approaches, but holds a considerable effort of maintaining the association between content and Web pages (Pröll et al. 1999, Labrinidis and Roussopoulos 2004). The same problems occur when using Web caching components to improve performance. Commercial caching tools like, e.g. SQUID (`http://www.squid.com`), allow for the setting of cache-control-directives, e.g., an expiry date, for each Web page controlling the update rate depending on the currency demand of a certain Web page (see also section 12.8.2).

6 There is no consistent use of content (data) quality dimensions. "Currency" (also termed "freshness"), generally describes how stale data is with respect to the data source and is often related to view consistency when materializing source data, while "timeliness" indicates whether data is out of date.

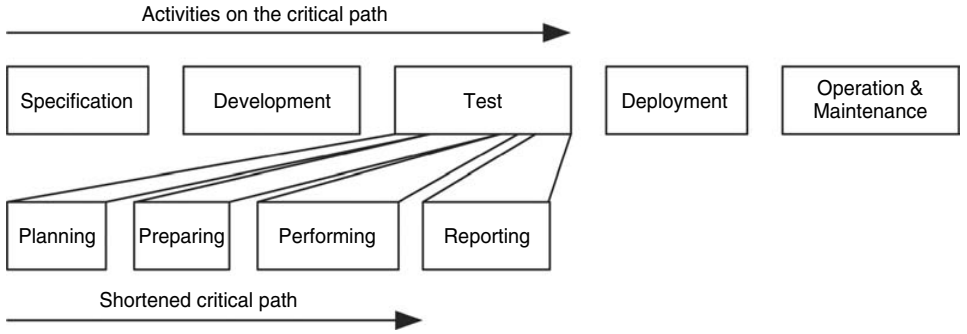

Figure 7-2 Critical path of activities.

roles but of close collaboration and best usage of the capabilities available in the team. This means that testing is an integrated development activity. The entire team is jointly responsible for the quality and thus for testing.

Agile approaches omit activities that don't seem to promise an immediate benefit. For example, they hardly document things or write test plans; instead, they communicate directly, clearly express expectations and jointly commit to meeting them. Team members have to cooperate closely and "understand" each other to ensure that errors are detected and analyzed quickly, and removed efficiently.

In an agile approach, the developers perform unit tests, i.e. they test their own work. By automating these unit tests, they can be used as small "change detectors". Whenever a small piece of functionality no longer works as previously, the change will be detected immediately. The delay between introduction of an error and detection is reduced significantly which typically makes it easier for developers to correct the error since recent activities or changes are still fresh in their minds. In addition to quick feedback, automated tests are an important prerequisite for short development cycles and for *refactoring* (redesigning a program while keeping the semantics to reduce redundancies and increase the design quality; see Fowler et al. 1999).

There may be a dedicated tester on the team who supports the developers and assumes the quality-assurance leadership within the team. Also, the tester may prepare functional tests (which are on a higher abstraction level than the developers' unit tests) and make test scripts tolerant to changes. In addition, the tester may support the customer with writing functional tests.

The following practices of Extreme Programming (XP) have a particular influence on testing and quality assurance.

- *Pair programming*: accelerates the exchange of knowledge between developers, between developers and testers, and generally within the team. Similar to software inspections, it also helps to detect errors early.
- *An on-site customer*: is available for questions with regard to the requirements at any time, and takes decisions in this respect. Together with the tester, the on-site customer prepares functional tests, which can also be used for acceptance tests later on.
- *Continuous integration*: ensures that small steps help minimize the risk of changes, and walks through all tests to continuously verify that the entire system is faultless.

- *Test-first development*: means that tests are written before the code, ensuring that the "developer pair" thinks about the "what" before it implements the "how". These tests are automated, so that they can be used for continuous integration.

It may be interesting to note that Feature-Driven Development (another agile approach, see Palmer and Felsing 2002), does not use Pair programming but rather promotes code reviews. Both approaches, however, guarantee that static quality assurance techniques are applied to the code right from the start.

The agile approach described here refers mainly to unit and acceptance tests. In contrast, conventional approaches are used for integration and system tests ("testing in the large"; see Ambler 1999).

7.5 Test Scheme

This section describes a generic scheme for Web application testing. The scheme merges the testing basics – test cases, quality characteristics, and test levels – described above into a uniform and manageable setting. The scheme represents a model for Web application testing designed to better understand how testing can be organized and to support a systematic, comprehensive, and risk-aware testing approach. In the form introduced here, the scheme can be used to visualize the aspects involved in testing, structure all tests, and serve as a communication vehicle for the team.

7.5.1 Three Test Dimensions

Every test has a defined goal, e.g., to check the correctness of an algorithm, to reveal security violations in a transaction, or to find style incompatibilities in a graphical representation. The goals are described by the required quality characteristics on the one hand – e.g., correctness, security, compatibility – and by the test objects on the other hand – e.g., algorithms, transactions, representations. Thus, quality characteristics and test objects are mutually orthogonal. They can be seen on two separate dimensions whereby the first dimension focuses on quality characteristics relevant for the system under test, and the second and orthogonal way to view testing is to focus on the features of the system under test. This viewpoint implies that the test objects are executed and analyzed during test runs, while the quality characteristics determine the objectives of the tests. Both dimensions are needed to specify a test and can be used to organize a set of related tests.

For a systematic testing approach it is useful to distinguish between these two dimensions so it will be possible to identify all the test objects affecting a certain quality characteristic or, vice versa, all the quality characteristics affecting a certain test object. This is important since not all quality characteristics are equally – or at all – relevant for all test objects. For example, a user of an online shop should be free to look around and browse through the product offer, without being bothered by security precautions such as authentication or encryption, unless the user is going to purchase an item. Hence, while the quality characteristic "security" plays a subordinate role for the browsing functionality of the shop, it is of major importance for payment transactions. Distinguishing between these two dimensions allows us to include the relevance of different quality characteristics for each single test object.

In addition, a third dimension specifies when or in what phase of the software lifecycle a combination of test object and quality characteristic should be tested. This dimension is

During operation of a Web application, care should be taken that the domains you own are managed carefully and the annual fees are paid punctually to prevent the risk of losing the domain due to sloppy handling. Continuous observation sometimes allows to buy previously taken domains that have become available. However, this marketing measure must not result in "domain squatting", i.e., registering a domain name without using it in the hope to cash it in later.

8.4 Content Management

"*Content Management* is the systematic and structured procurement, creation, preparation, management, presentation, processing, publication, and reuse of contents" (Rothfuss and Ried 2001). This definition can be applied to the contents of a Web application, e.g., text, images, audio, video, etc., represented in HTML documents, which is then known as *Web content management* (Jablonski and Meiler 2002).[5] The content's quality is a particularly important factor for the acceptance of Web applications (see Chapter 1). Up-to-date and rich Web application content serves to win customers, but also particularly to increase site stickiness, i.e., anything about a Web site that encourages a visitor to stay longer and to return. In this respect, a Web application should have a critical minimum mass of high-quality content at the time when it is launched. However, the highest cost for creating and preparing content arises mainly during the period following the Web application's launch, thus forming a central task of the operation and maintenance phase.

From the *organizational viewpoint*, appropriate measures have to be taken, such as the forming of an independent content editing group, including a definition of the workflows and an appropriate role distribution of the people participating, or outsourcing ongoing content maintenance to a third-party service provider. The use of content syndication requires appropriate contract agreements (see section 8.4.2). Another strategic aspect relates to copyright, authors' rights, etc. Particular attention should be paid to legal issues in connection with the rights for images, videos, logos, etc., which are to be used in a Web application. Readers can find a wealth of information about copyrights on the Internet; visit, for example, (`http://www.copyright.gov`).

From the *technical viewpoint*, it is important that the Web application's architecture allows to farm out content management to non-technical staff in the sense of separating content, structure, and presentation. Among other things, this concerns questions as to whether or not a commercial tool, i.e. a content management system (CMS), should be used, or whether system components oriented to content management and reasonably based on a database management system should be implemented (see also sections 4.6 and 5.3).

Another important aspect of content management is the Web's *globality*, which means that pages may have to offer *multilingual content*, which normally translates into considerable cost. In any event, content should be prepared in the national language of a site's main target group and additionally in English. Consideration of the number of language versions is important as early as in the design phase and should be made with regard to the maintenance cost. In most cases, however, it does not pay to build internal foreign-language competencies. Cooperation with a translation agency or freelance translators is normally a better choice. A related aspect are *currency* issues, which are of interest in content syndication implementations (see section 8.4.2).

5 Note that (Web) content management is a buzz-term not used consistently by the industry (`http://www.gilbane.com/gilbane_report.pl/6/What_is_Content_Management`).

a Web application owner should try to both set links in his or her Web application and see that reciprocal linking is observed by other (popular) Web sites.

Moreover, to improve the quality of search engines' results, Web application developers should make extensive and correct use of other meta-tags, e.g., the *meta-tag content*, the *meta-tag date*, or the *meta-tag content-language*. Web application developers should also consider to use the Dublin Core Metadata (`http://dublincore.org/`).

Another interesting aspect for Web application developers are options to exclude search engines from certain pages of their Web application, or exclude certain search engines from visiting the Web application entirely or in part. The former option is meaningful, for example, if a "visitors counter" should measure only actual visitors, but no robot accesses. The second option is useful, for example, if some parts of a Web application contain no information relevant for search engine users. To set such options, we can use the (`robots.txt`) file or the "ROBOTS" meta-tag (`http://www.robotstxt.org/wc/exclusion.html`).

8.3.4 Content-related Marketing

The absolute pioneer with regard to the increasingly popular advertising form of content-related marketing is the Google search engine. In fact, Google developed a special program called AdWords (`https://adwords.google.com`) to reach potential customers in a targeted way. Building on their successful search engine technology for targeted use of keywords, this new program additionally allows you to place matching ads both on search result pages and on pages of affiliate Web sites. AdWords campaigns are used to reach users looking for special keywords within Google, and, moreover, put ads on related Web applications.

This content-related marketing extends the distribution of AdWords far beyond the huge Google network of search pages. For example, if users display the weather forecast for Munich on a weather Web page, they might concurrently see ads for hotel rooms and rental cars in the greater Munich area. The benefit of content-related marketing for users is that contents with corresponding products and services are linked. As a result, advertisers can reach more customers on a large number of new Web sites on the Internet.

8.3.5 Domain Management

Each Web application has an Internet address which is used for unique and direct access to that Web application. In addition, this address has an important strategic significance for marketing purposes, as mentioned in section 8.3.3. Today, registering is offered by a large number of domain registration companies responsible for assigning and managing domains and operating on national and global levels. It is recommended to use only one single domain to communicate with the outside world. This so-called "master domain" can be thought of as a digital brand of the Web application owner. In addition, it is recommended that other domains, which relate to the master domain and its products and services, are used as secondary domains which also lead to the Web application. This ensures that competitors cannot obtain these domains and use them for their own purposes. In addition, a well-conceived domain concept is a greatly beneficial "traffic generator".

necessary to describe the timeframe within which the testing activities take place: from early phases such as requirements definition over design, implementation, and installation to operation and maintenance. As a result, testing can profit from valuable synergies when taking the activities over the whole lifecycle into account, e.g., by designing system tests that can be reused for regression testing or system monitoring. Furthermore, the time dimension helps to establish a general view of testing over all phases and allows a better understanding of what effects quality characteristics and test objects over time. It makes it easier to justify investments in testing in early phases since the possible payoff in later phases becomes clear.

If we join these three dimensions – *quality characteristics, test objects*, and *phases* – the result can be visualized as a three-dimensional cube as shown in Figure 7-3 (see Ramler et al. 2002). The cube contains all tests as nodes at the intersection of a specific quality characteristic, test object, and phase. The figure shows a possible structuring for the three dimensions, as suggested for Web application testing in the next section.

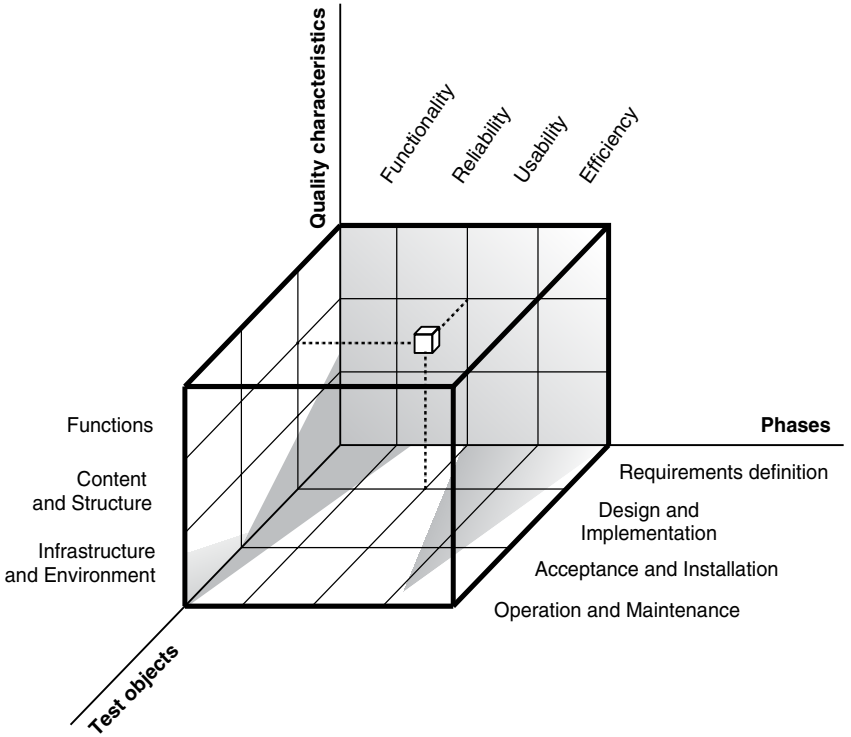

Figure 7-3 Test scheme for Web applications (Ramler et al. 2002).

7.5.2 Applying the Scheme to Web Applications

This section describes how the dimensions of the generic scheme introduced in the previous section can be structured to accommodate the special characteristics of Web applications and

Web projects. In practice the structuring depends on the requirements of the system under test. Therefore, it is necessary to customize and detail the generic scheme according to the specific situation of the project.

Quality Characteristics

The *quality characteristics* dimension is determined by the quality characteristics that are relevant for the Web application under test. Thus, the quality characteristics relevant for testing originate in the objectives and expectations of the stakeholders and should have been described as non-functional requirements in the requirements definition (see Chapter 2). Additional information from other quality assurance measures and testing experience (e.g., typical risk factors, common exploits) should be considered when defining the quality characteristics dimension for a specific Web application.

For a generic classification we suggest to use the quality characteristics proposed by the ISO/IEC 9126-1 standard, respectively a representative subset such as *functionality, reliability, usability*, and *efficiency* (Olsina et al. 2001). A further breakdown results from the hierarchy of characteristics and sub-characteristics specified in the standard.

Test Objects

Traditional software testing describes the *test objects* dimension mainly by the functions of the system under test, specified in the form of functional requirements. The software testing literature (e.g. Myers 1979, Beizer 1990, Jorgensen 2002) elaborates on the design of test cases based on functional requirements at great length.

In contrast to conventional software systems, which focus mainly on functional requirements, Web applications also provide content, which frequently has to be developed and tested as part of a Web project (Powell et al. 1998). In document-centric Web applications, the content is often as important for their users as the actual functionality. For testing, this translates into the requirement to detect errors in the content including the hypertext structure and the presentation.

Whether or not a Web application meets users' expectations under real-world conditions depends also on the infrastructure and the environment of that Web application. This includes, for example, Web server configuration, network connection, integrated partner companies, and associated workflows. Users are confronted with the entire system. From the users' perspective, it doesn't really matter whether a Web application falls short of meeting their expectations due to a programming error or due to faulty Web server configuration. For this reason, it is necessary to extend testing to the infrastructure and the environment of a Web application.

The generic test objects dimension should, therefore, include the *content and structure* and the *infrastructure and environment* of a Web application, in addition to *functions*.

Phases

The third dimension of the scheme – *phases* – focuses on the temporal sequence of Web application testing. This dimension shows when which test should be run within a Web application's lifecycle. This third dimension is structured according to a general development

marketing aspect. Notice that there is normally no way to guarantee or force the listing or ranking of a Web application among the first set of, say, ten entries. One reason is that there is no general detailed knowledge of how each of the popular search engines calculates the ranking of search results. Another reason is that their strategies change constantly. Yet another reason is that many competitors on the Web equally try to achieve top ranking. But Web application owners can consider a few "rules" to try to achieve a good ranking for their Web applications. To this end, Web application owners should put themselves in a searching user's place and define the keywords users might use to find their Web applications. These keywords should appear in certain positions within a Web application, where care should be taken that robots will still be able to acquire data invisible for users of search engines, or even prefer them with regard to Web application ranking (for example, meta-tags; see also `http://www.searchenginewatch.com/webmasters/article.php/2167931`). In the following we list a few specific measures:

- *The Web application URL*: The domain of a Web application should ideally contain descriptive keywords, e.g., (`http://www.travel.com`) for a travel-industry Web application.
- *The "title" meta-tag*: In addition to indexing, some search engines output the words in the title of a Web page as additional information in the result list.
- *The "keywords" meta-tag*: Many but not all search engines use the terms listed here as preferred words in indexing, which means that a Web application can achieve a better ranking when users search for one of these keywords. The Web application owner should state as many keywords as possible that describe the topics of his or her Web application, for example, `<META name="keywords"content="travel, Austria, mountain, lake, hiking, wellness,...">`. But notice that some search engines limit the length (e.g., 1000 bytes). Moreover, care should be taken that, though repeating the same words can improve the ranking, repeating the same words excessively to improve the ranking is seen as "spamming" and "punished" by some search engines by placing the Web application down the list or not listing it. The same applies to "spoofing", i.e., intentionally setting false information (e.g., using "Hawaii" as a keyword for a Web application that sells used cars).
- *The "description" meta-tag*: Some search engines also use the description you supply as a priority in their search process.
- *The Web page content*: Since some search engines use only the first x bytes of a Web page rather than its full content for indexing purposes, its beginning should include a semantically meaningful textual description rather than images, tables, etc. When using images, there should be an additional textual description as an alternative. In addition, it should be borne in mind that robots are unable to "fill out" forms, so that alternative navigation at aids to other pages should be provided.
- *Link popularity and PageRank*: One measure that has become very popular, especially for the scientific publications of the founders of the Google search engine, is to increase the link popularity, i.e., increasing the number of links that point to a Web application. More specifically, a "PageRank" calculated for each Web page on the Web is used to calculate the ranking based on this rule: "A document is the more important the more often it is linked to in other (popular) documents" (Brin and Page 1998).[4] To increase this PageRank,

4 The corresponding algorithm has been trademarked as PageRank™ (cf. `http://www.google.com/technology/`).

McGill 1983, Baeza-Yates and Ribeiro-Neto 1999). In this environment, documents are acquired by creating a representation of them, generally in the form of an *index* describing the content and semantics of a document. In general, this technique filters semantically meaningful words (terms) from a document, puts them into a uniform "stemmed" form (e.g., "golfers" changes to "golf"), and assigns a weight to each term, e.g., according to its frequency of occurrence in the document, or where it is positioned within a document. When a user requests one or several terms, the system calculates relevant documents based on certain *retrieval models*, and presents them in an *ordered result list*, sorted in descending order by the probability of their relevance. In addition to the occurrence of the terms searched in a document, these retrieval models also consider weights and other factors in the relevance calculation.

Basically the same approach applies to current search engines on the Web, both with regard to Web document acquisition – Web pages uniquely specified by URLs – and processing of user requests. However, Web specifics like the distribution aspect of Web documents across networks that are accessed differently, heterogeneous data formats, and the Web's volatile character with regard to the uncontrolled growth and the uncontrolled change of existing content, have to be taken into account.

To acquire Web documents and to permanently update the data repository of a search engine, so-called *robots*, also known as *spiders* or *crawlers*, are used. Beginning on a start page, a robot follows links to visit new or modified Web pages and subsequently downloads them for indexing purposes (Lawrence and Giles 1998, Sullivan 2002).

Based on this background knowledge, the following questions arise in view of conducting search engine marketing:

- What can I do to make sure my Web application *gets listed* in (major) search engines?
- What can I do to make sure my Web application gets *a top ranking* in (major) search engines?

In the following, essential measures to meet these requirements are discussed. Further tips and tricks on search engine marketing can be found at (`http://searchenginewatch.com`).

Search Engine Submission

Unless relying on a robot to accidentally find a Web application, the URL of the homepage of a Web application should be manually submitted to selected search engines. All popular search engines have registration forms on their Web sites, allowing to enter a short description of the Web application in terms of approximately 25 words with two or three expressive key terms. The processing time of a registration normally takes several weeks.

In addition, good embedding into the Web, i.e., being linked to in other – ideally prominent – Web sites or access points can influence how easily robots find a Web application. In this sense, so-called "reciprocal linking", i.e., linking from other Web sites to the own Web application (see also section 8.3.2), is a meaningful marketing measure not only to create direct user traffic, but also to draw robots' attention to a Web application.

Search Engine Ranking

The majority of users look only at the first 10 to 20 entries in a search engine's result list, which means that good ranking (also termed placement) of a Web application is an essential

process or software lifecycle. The phases of a development process can differ from one Web project to another, depending on the selected process model (see Chapter 10). As a general rule, it is sufficient for a simple generic categorization to roughly distinguish between the following phases: *requirements definition, design and implementation, acceptance and installation*, and *operation and maintenance*.

The inclusion of all phases of the entire lifecycle makes clear that testing of a Web application doesn't end when a project is completed. For example, monitoring repeats a part of the tests regularly in normal operation to find new or existing errors after changes have been made to the infrastructure or to the environment. Typical examples for such changes are Web server updates or new Web browser versions. For instance, we have to periodically run compatibility tests to assure that users can access a Web application with any Web browser as soon as a new version becomes available, although no changes were made to the Web application itself.

7.5.3 Examples of Using the Test Scheme

To handle the three dimensions of the cube presented in the previous section, we use two-dimensional matrices representing slices or projections to remove one of the three dimensions. An example is Table 7-1, which shows the two dimensions test objects and quality characteristics. Thereby the scheme is used as an overview and conceptual framework to systematically arrange the methods and techniques applicable for Web application testing. The matrix can be used to establish a project-specific or company-wide method and tool box for Web application testing.

A further example is presented in (Ramler et al. 2002) which shows the application of the scheme in risk-based test management to prioritize tests for a Web application. The testing priorities have been defined in a joint workshop with testers, developers, users, and domain experts based on the priorities of the use cases (test objects) and non-functional requirements (quality characteristics). This approach facilitates test planning and test effort estimation and allows to trace back the priorities of each test to the priorities of the requirements.

7.6 Test Methods and Techniques

When testing Web applications, we can basically apply all methods and techniques commonly used in traditional software testing (see Myers 1979, Beizer 1990, Kaner et al. 1999, Jorgensen 2002). To take the specifics of Web applications into account, some of these test methods and techniques will have to be thought over, or adapted and expanded (e.g., "What influence factors have to be taken into account when testing compatibility with different Web browsers?"). In addition, we will most likely need new test methods and techniques to cover all those characteristics that have no correspondence in traditional software testing (e.g., testing of the hypertext structure).

The summary shown in Table 7-1 corresponds to the test scheme introduced in section 7.5 and is structured by the *test objects* dimension and the *quality characteristics* dimension. The table (see also Ramler et al. 2002) gives an exemplary overview of the methods, techniques, and tool classes for Web application testing described in the literature (e.g., Ash 2003, Dustin et al. 2002, Nguyen et al. 2003, Pressman 2005, Splaine and Jaskiel 2001). It shows typical representatives of test methods and techniques as a basis for arranging a corporate or project-specific method and tool box.

Table 7-1 Methods, techniques, and tool classes for Web Application testing

		Functions	Content and Structure	Infrastructure and Environment
Functionality	**Suitability**	Reviews and inspections, Test-driven development	Checklists, Lexical testing, Style guides, Reviews	
	Accuracy	Capture/Replay, Test-driven development	Static analysis, Link testing, Lexical testing, Reviews	Static analysis, Link testing
	Interoperability	Cross-browser and cross-platform compatibility testing	Test printing, Checklists, Reviews, Compatibility testing	Cross-browser and cross-platform compatibility testing
	Compliance	Compatibility testing, Style guides, Test-driven development	Checklists, Compatibility testing, Style guides, Reviews	Cross-browser and cross-platform compatibility testing
	Security	Analysis of common attacks, Reviews and inspections		Analysis of common attacks, Forced-error testing, Ethical hacking
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	Static readability analysis, Usability studies	
	Learnability	Usability studies, Heuristic evaluation		
	Operability	Usability studies, Heuristic evaluation		Heuristic evaluation
	Attractiveness		Publicity testing	
Efficiency	**Timing Behavior**	Load and Stress testing,Monitoring		Load and Stress testing, Monitoring
	Resource Utilization	Endurance testing	Load testing	Endurance testing, Monitoring

The integration options are manifold, reaching from a simple text link or a product presentation containing a link, to advanced shop modules, which map the entire process chain of the Web application and implement the representation or user guidance in the look and feel of the affiliate site.

Most revenue agreements are based on a pay-per-performance model, a Web-based business model, which collects a fee only if the consumer completes a transaction (Hoffmann and Nowak 2005, Rappa 2003). Variations include:

- Banner exchange
- Pay per click (affiliate is paid for a user click-trough)
- Revenue sharing (percent-of-sale commission).

Naturally, embedding your own Web application in an issue-related environment promises more acceptance and supports sales proactively. The most important success factors of affiliate marketing can be described as follows:

- The integration into an affiliate site should not only be based on simple links or banners, but should provide added value to users. Creating a so-called "customized version" (persistent navigation and look and feel) with clear business terms and conditions for visitors normally increases site traffic and sales.
- A strong affinity of topics between the affiliate sites increases the interest of users.
- When preparing the contract, care should be taken to reach a win-win situation. Fair commercial conditions, increasing the popularity or the traffic, are important indicators and create a sound basis for long-term cooperation between the affiliates.

Even if affiliate marketing primarily concerns the company's marketing team, developers are requested to, for example, care for the seamless integration into the Web application, or for a proper handling of reliability problems of the affiliate's Web application.

8.3.3 Search Engine Marketing

Among the most popular services on the Web are search engines like the ones based on Crawlers, i.e., Google (`http://www.google.com`) and AllTheWeb (`http://www.alltheweb.com`), or the mostly manually created catalog Yahoo (`http://www.yahoo.com`). By trying to "forward" users with a specific information need as target-specific as possible to relevant Web pages, search engines represent potentially long-range online services to mediate relevant contacts.[3]

Search engine marketing (also termed search engine promotion) attempts to win relevant contacts by appearing on the search result pages of these services. These placements can be in the form of a response to a user's request within the search result list, or in the form of ads, or by the use of content syndication (see section 8.3.4 and section 8.4.2). We will look at the first of these applications more closely below.

For search engine marketing to be successful, we need to know how search engines work. Search engines are based on the concepts of the conventional *information retrieval systems* of the 1960s, which were aimed at storing and retrieving textual data repositories (Salton and

3 According to a study conducted by the Web analysis company WebSideStory in June 2005, Google's U.S. search referral percentage hit exceeded 52% and is even higher in European countries, e.g., more then 90% in Germany.

issue is often neglected, mostly for know-how or budget reasons, though specialists like Web designers, developers, and marketing experts agree that it is of extreme importance to invest in targeted demand management and Web application promotion. Traditional advertising channels like newspaper ads, TV or radio commercials, posters, business print matters, direct marketing, etc., are not the only media we can use to announce and promote Web applications; special focus should be placed on *webvertising*, i.e., using Web-based methods to advertise in the Web. The following sections will discuss some of the common webvertising methods.

8.3.1 Newsletters

Newsletters, which are generally distributed by e-mail, are an effective medium to win new customers, and to provide assistance to the existing customer base. A large number of companies increasingly use newsletters to create strong relationships with existing or potential customers. However, advertising messages and self-portraits should be kept on modest levels, and users should be offered "neutral" and value-adding content, e.g., information about new products. A few proven rules should be observed when creating a newsletter, e.g., clear content orientation, content currency, regular dispatch, frequent distribution list updates, unsubscribe option, etc. Many sources on the Internet provide useful hints for newsletter publishers, e.g., (`http://companynewsletters.com/indexonline.htm`).

From the developer's perspective, apart from integrating a newsletter component into the overall architecture, appropriate interfaces for automated acquisition of news, products, etc., and perhaps an interconnection with customization components, should be designed. For example, for a travel-industry Web application, when creating a last-minute offer, users who have shown interest in certain products by customizing things could automatically be informed via a newsletter. In addition to brief information about the new product, the newsletter should include a link that points directly to the appropriate page of the Web application.

8.3.2 Affiliate Marketing

There is a popular myth about the origins of affiliate marketing. Legend has it that Jeff Bezos, founder of Amazon.com, chatted with a woman at a cocktail party about how she wanted to sell books about divorce on her Web site. After that exchange, Bezos pondered the idea and thought about having the woman link her site to Amazon.com and receive a commission on the book sales. This is believed to have been the impetus for creating the first associates program on the Web.[2] Today, the affiliate marketing idea is one of the most important factors for Amazon's popularity and success.

Affiliate marketing means the revenue sharing between online advertisers/merchants and online publishers/sales people in the way that products, services, or functionalities of somebody's Web application are integrated into an affiliate's Web application. Though affiliate marketing is often tied to the loss of the own identity, it offers an efficient form of advertising at relatively low startup cost to attract more visitors to one's own Web pages and to make the Web site popular.

2 However, there were many adult sites that dabbled in the affiliate marketing concept before Amazon.com picked it up – cf. (`http://www.affiliatemanager.net/article17.shtml`).

The following subsections briefly describe typical methods and techniques for Web application testing.

7.6.1 Link Testing

Links within a hypertext navigation structure that point to a non-existing node (pages, images, etc.) or anchor are called *broken links* and represent well-known and frequently occurring errors in Web applications. To test for correct linking of pages (*link checking*), all links are systematically followed beginning on a start page, and then grouped in a link graph (*site map*).

When running a link checking routine, one usually finds not only links that point to non-existing pages, but also pages which are not interlinked with others or so-called orphan pages. An *orphan page* can be reached via a link, but doesn't have a link back to the hypertext structure. To casual users it is not obvious where to go next, so they abandon the website. Pages are ideally designed so that they end with a suggestion of where the reader might go next.

In addition, when traversing links, one can often find additional data that supply indications to potential errors, e.g., the depth and breadth of the navigation structure, the distance between two related pages, measured by the number of links, or the load times of pages.

7.6.2 Browser Testing

A large number of different Web browsers can be used as the client for Web applications. Depending on the manufacturer (e.g., Microsoft, Mozilla, Netscape, Opera), or the version (e.g., Internet Explorer 5.0, 5.01, 5.5, 6.0), or the operating system (e.g., Internet Explorer for Windows XP/2000, Windows 98/ME/NT, or Macintosh), or the hardware equipment (e.g., screen resolution and color depth), or the configuration (e.g., activation of cookies, script languages, stylesheets), each Web browser shows a different behavior. Standards like the ones specified by W3C are often not fully implemented and "enhanced" by incompatible vendor-specific expansions. Web browser statistics and settings are available online (e.g., at `http://www.webreference.com/stats/browser.html`).

Browser testing tries to discover errors in Web applications caused by incompatibilities between different Web browsers. To this end, one normally defines a Web application's core functions, designs suitable test cases, and runs the tests on different target systems with different browser versions. During these tests, one should ask the following questions:

- Is the Web application's state managed correctly, or could inconsistent states occur when navigating directly to a page, for example, by using the browser's "Back" button?
- Can a (dynamically generated) Web page be bookmarked during a transaction, and can users navigate to that page later without having to enter a user name and password to log in?
- Can users use the Web application to open it in several browser windows (one or several instances of the Web browser) concurrently?
- How does the Web application react when the browser has cookies or script languages deactivated?

To limit the number of possible combinations of browsers, platforms, settings, and various other influence factors to a manageable set of test cases, the configurations of existing or potential

users need to be analyzed, e.g., by evaluating log files and consulting browser statistics, to find popular combinations.

7.6.3 Usability Testing

Usability testing evaluates the ease-of-use issues of different Web designs, overall layout, and navigations (see Chapter 11) of a Web application by a set of representative users. The focus is on the appearance and usability. A formal usability test is usually conducted in a laboratory setting, using workrooms fitted with one-way glass, video cameras, and a recording station. Both quantitative and qualitative data are gathered.

The second type of usability evaluation is a heuristic review. A heuristic review involves one or more human-interface specialists applying a set of guidelines to gauge the solution's usability, pinpoint areas for remediation, and provide recommendations for design change. This systematic evaluation employs usability principles that should be followed by all user interface designers such as error prevention, provision of feedback and consistency, etc.

In the context of usability testing the issue of making the Web accessible for users with disabilities has to be treated (see Chapter 11). Accessibility means that people with disabilities (e.g., visual, auditory, or cognitive) can perceive, understand, navigate, and interact with the Web. The Web Accessibility Initiative (WAI) of the W3C has developed approaches for evaluating Web sites for accessibility, which are also relevant for testing Web applications. In addition to evaluation guidelines the W3C provides a validation service (`http://validator.w3.org/`) to be used in combination with manual and user testing of accessibility features.

7.6.4 Load, Stress, and Continuous Testing

Load tests, stress tests, and continuous testing are based on similar procedures. Several requests are sent to the Web application under test concurrently by simulated users to measure response times and throughput. The requests used in these tests are generated by one or several "load generators". A control application distributes the test scripts across the load generators; it also synchronizes the test run, and collects the test results.

However, load tests, stress tests, and continuous testing have different test objectives:

- A *load test* verifies whether or not the system meets the required response times and the required throughput (see also Chapter 12). To this end, we first determine load profiles (what access types, how many visits per day, at what peak times, how many visits per session, how many transactions per session, etc.) and the transaction mix (which functions shall be executed with which percentage). Next, we determine the target values for response times and throughput (in normal operation and at peak times, for simple or complex accesses, with minimum, maximum, and average values). Subsequently, we run the tests, generating the workload with the transaction mix defined in the load profile, and measure the response times and the throughput. The results are evaluated, and potential bottlenecks are identified.
- A *stress test* verifies whether or not the system reacts in a controlled way in "stress situations". Stress situations are simulated by applying extreme conditions, such as unrealistic overload, or heavily fluctuating load. The test is aimed at finding out whether

availability and an *unlimited number of users*, causes considerable cost in terms of budget for hardware/software equipment as well as personal effort. We can observe again and again that budgets for normal usage have not at all been considered in the planning of Web applications, or only insufficiently (see also section 9.4.3 for project risks). This situation can become very critical since the update cycles are much shorter compared with non-Web applications and should be included in the budget planning. An executive survey conducted by Jupiter Research in October 2003[1] showed that more than 58% of Web sites planned a relaunch in 2004. The typical 24x7 operation makes it unavoidable to deal with issues relating to the technical operation as early as in the conception phase; and they should be planned and implemented in close cooperation with the IT department, if existing. Basically, there are two strategic options to be considered, similarly to how it is done for non-Web applications: outsourcing or internal operation.

Another important factor for a Web application's success or failure is the embedding of the project into the workflow of the company or organization concerned. Professional planning will show that in many cases a *redefinition* of *workflows* relating directly or indirectly to the Web application has to be conducted. This is often the only way to increase productivity, to reduce cycles, and to improve quality assurance. In this context, *helpdesk services* have proven useful, especially for commercial Web applications. A helpdesk operated by members of staff can be contacted by the Web application's users via phone or e-mail. Depending on the Web application type and orientation towards interests, inquiries received by e-mail are normally answered free of charge. Information provided by phone is typically financed by value-added services, i.e., users are charged per phone minute. For e-commerce applications, *service or call center support* can be offered to users to increase their "buying inclination" via the implementation of a call-back function. For example, travel-industry platforms could offer information and booking support by specially trained employees who guide users through the electronic booking process to help them find matching packages.

In general, it cannot be emphasized enough that *unlimited side conditions, competitive pressure*, and *short lifecycles* require permanent further development of a Web application during its normal operation. Once a Web application has been launched, errors will occur during normal operation, environment conditions (new system software, new hardware) change frequently, and new user wishes and requirements with regard to contents, functionalities, user interface, or performance, will emerge. Quality assurance measures such as forming an interdisciplinary project team are often omitted, so that the need for re-engineering work during the maintenance and operation phase increases out of proportion. Web applications are usually outdated after only a few months unless they are continually improved and errors permanently removed. Otherwise, the consequences can be that not only the Web application's original purpose is at stake, but, moreover, the image and credibility of its owner may suffer due to the wide public presence of Web applications.

The following section discusses how a Web application can be promoted, what things have to be observed in content management, and how a Web application's usage can be analyzed.

8.3 Promoting a Web Application

"How will users find my Web application and its information?" Answers to this and similar questions are very important, especially for commercial Web applications. Nevertheless, this

1 `http://www.websage.net/blog/archives/web_site_optimization/000043.html`.

is often heavily influenced by external consultants and software vendors. In particular, issues concerning ongoing maintenance and operation are seldom discussed or specifically addressed. The impact is often fatal, because Web applications pose special requirements on maintenance and operation.

Time-to-market pressure tempts people to reduce the conception and development cycles, which results in sloppy development work. The consequences are that products are faulty, poorly performing, unreliable, poorly scalable, and seldom user-friendly. This means that actual development tasks are delegated to the maintenance and operation phase. On the other hand, rapidly walking through the development cycles allows people to collect immediate feedback on requirements from the "cyber world". This shows not only problems and errors, but also allows Web application developers to build in new functions, which hadn't been anticipated in the conception phase. This leads to the situation that development and maintenance are eventually intertwined.

This chapter takes a practical view discussing selected organizational and developmental aspects which mainly fall into the phase after the launch of the Web application conventionally named the *operation and maintenance phase*, but also referred to as the *deployment phase*. We will look at three of these aspects more closely, because they experience a totally new quality, especially in the context of the Web. Section 8.3 deals with *measures to promote* a Web application, in particular how to get a Web application listed in search engines. Section 8.4 is dedicated to the issue of *managing content* and how maintenance work can be automated. Requirements on further development can be derived, among other things, from *analyzing the accesses* to a Web application, and analytical techniques will be discussed in section 8.5. The following section summarizes miscellaneous operation and maintenance tasks which were mainly deduced from Web application specifics.

8.2 Challenges Following the Launch of a Web Application

After the Web application is installed and put into practical use, activities to maintain normal operation have to be conducted. Also, corrective, adaptive, and perfective measures (Sommerville 2004) have to be taken.

As mentioned earlier, the phase that deploys, operates, and maintains Web applications is often underestimated in the planning phase, although it poses more diverse challenges to the operators and developers, compared with traditional software development. The reason is found in some of the specifics of Web applications discussed in Chapter 1. It should be noted that there are major differences between the global usage of an application on the Internet versus its use by a restricted user group, i.e., in an intranet or extranet. Though intranet and extranet applications are based on the same technologies as globally available Web applications, they are similar to non-Web applications with regard to their usage, e.g., training possibilities, technical operation, and promotion (see also Chapter 1), so that they are not a major issue in this chapter.

One of the most important characteristics of a Web application's deployment phase is that it becomes *immediately and globally available*. In many cases, the first-time introduction of a new Web application entails considerable adjustments and related problems. Factors like market demands and customer interests, which usually emerge to their full extent only after a Web application has been deployed, have to be brought to a common denominator. For Web applications, the operation of the IT infrastructure, which typically has to stand up to *24x7*

or not the system reaches the required response times and the required throughput under stress at any given time, and whether it responds appropriately by generating an error message (e.g., by rejecting all further requests as soon as a pre-defined "flooding threshold" is reached). The application should not crash under stress due to additional requests. Once a stress situation is over, the system should recover as fast as possible and reassume normal behavior.

- *Continuous testing* means that the system is exercised over a lengthy period of time to discover "insidious" errors. Problems in resource management such as unreleased database connections or "memory leaks" are a typical example. They occur when an operation allocates resources (e.g., main memory, file handles, or database connections) but doesn't release them when it ends. If we call the faulty operation in a "normal" test a few times, we won't detect the error. Only continuous testing can ensure that the operation is executed repeatedly over long periods of time to eventually reproduce the resource bottleneck caused by this error, e.g., running out of memory.

7.6.5 Testing Security

Probably the most critical criterion for a Web application is that of security. The need to regulate access to information, to verify user identities, and to encrypt confidential information is of paramount importance. Security testing is a wide field, and will be discussed in this section only briefly; it does not represent a testing technique in the literal sense. It concerns issues in relation to the quality characteristic "security":

- *Confidentiality*: Who may access which data? Who may modify and delete data?
- *Authorization*: How and where are access rights managed? Are data encrypted at all? How are data encrypted?
- *Authentication*: How do users or servers authenticate themselves?
- *Accountability*: How are accesses logged?
- *Integrity*: How is information protected from being changed during transmission?

When testing in the field of security, it is important to proceed according to a systematic test scheme (see section 7.5). All functions have to be tested with regard to the security quality characteristic, i.e., we have to test each function as to whether or not it meets each of the requirements listed above. Testing security mechanisms (e.g., encryption) for correctness only is not sufficient. Despite a correctly implemented encryption algorithm, a search function, for example, could display confidential data on the result page. This is an error that test runs should detect, too. Typically, security testing must not only find defects due to intended but incomplete or incorrect functionality but also due to additional yet unwanted behavior that may have unforeseen side-effects or even contains malicious code. Unwanted, additional behavior is often exposed by passing input data unexpectedly to an application, e.g., by circumventing client-side input validation (Offutt et al. 2004).

A large amount of information on typical security holes in Web applications is available in the Internet (e.g., common vulnerabilities and exposures are discussed at `http://www.cve.mitre.org`; a checklist of security issues is available at `http://www.w3.org/Security/Faq/www-security-faq.html`). Chapter 13 of this book is dedicated to security issues.

7.6.6 Test-driven Development

Test-driven development (Beck 2002) emerged from the test-first approach used in Extreme Programming, but it does not necessarily dictate an agile project approach. This means that we can use this technique even in conventional projects.

As the name implies, test-driven development is driven by (automated) tests, which are created prior to coding work. New code is written if a previously created test fails, i.e. developers have to write tests before they proceed to the implementation (refactoring). In that way, the design (and consequently the application) grows "organically" and every unit has its unit tests. The design naturally consists of many highly cohesive and loosely coupled components, facilitating the test.

Once the test fails, the developer implements what is absolutely necessary to successfully run the test as quickly as possible, even though this may mean violating a few principles. Once in a while, the developer eliminates the duplicate code introduced during the implementation. The many small unit tests can work as small "change detectors" during the course of the project.

Test-driven development has a beneficial psychological effect; the developer can concentrate on small steps and keep the larger goal ("clean code that works") in mind. This is opposed to the typical vicious circle; if increased pressure has left less time for testing, so that fewer things are tested, then more uncertainties lead to more pressure. Test-driven development ensures that a developer under increased stress simply runs existing automated tests more often. This enables him or her to get direct feedback that things are still working, which reduces stress and error probability.

7.7 Test Automation

Testing of large systems greatly benefits from tools that implement and automate methods and techniques (see above). This holds true particularly for iterative and evolutionary development of Web applications where an organized use of tools can support tests that are repeated frequently within short development cycles and narrow timeframes. But even once development is completed, changes to the infrastructure and the environment of a Web application often require tests to be repeated.

7.7.1 Benefits and Drawbacks of Automated Tests

Automation can significantly increase the efficiency of testing and, furthermore, enables new types of tests that also increase the scope (e.g. different test objects and quality characteristics) and depth of testing (e.g. large amounts and combinations of input data). Test automation brings the following benefits to Web application testing (see also Fewster and Graham 1999):

- Running automated regression tests on new versions of a Web application allows to detect defects caused by side-effects to unchanged functionality. These regression tests help to protect existing functionality of frequently changing Web applications.
- Various test methods and techniques would be difficult or impossible to perform manually. For example, load and stress testing requires automation and corresponding tools to simulate a large number of concurrent users. In the same way it is virtually impossible to fully test all the links of a Web application's extensive, cyclic hypertext structure manually.

8 Operation and Maintenance of Web Applications

Arno Ebner, Birgit Pröll, Hannes Werthner

The serious side of the digital life of a Web application begins on the day it is launched – when it enters its operation and maintenance phase. While this phase is rather clearly delimited from the development phase in conventional applications, Web applications experience *de facto* a merge with operation and maintenance. This fact results particularly from a number of tasks that cannot be done separately from the development phase, since they have a strong influence on the development.

This chapter cannot exhaustively discuss the operation and maintenance issues of Web applications because, especially in the field of Web Engineering, no methodological approaches are available yet. Instead, we will pick out three major issues to show specifics in operating and maintaining Web applications, compared with conventional applications. We will first discuss how Web applications can be announced and promoted by taking measures which have a decisive impact on how a Web application appears in search engines or on affiliate Web sites. These measures have, in turn, a strong influence on the development of a Web application, e.g., by setting meta-tags, or by specifically choosing the link structure. Another focus of this chapter is the management of a Web application's content. Depending on content currency and performance requirements, Web pages are either pre-generated or created on the fly. The reuse of content from third-party vendors is supported by content syndication. And finally, we will deal with the usage analysis issue. One of the objectives of usage analysis is to identify improvement potential in a Web application by measuring and evaluating user accesses to the Web application to identify user behavior patterns. Improvement measures that can be derived from these statistics include not only the optimization of a Web application's usability, but also an increase of its performance by, for example, pre-fetching popular Web pages.

8.1 Introduction

Within the scope of designing, planning, and implementing Web applications, project management often fails to consequently orient to the strategy, structure, culture, processes, and business idea of a company or organization (see also Chapter 9). Under the motto "Let's have an online presence", many companies or organizations proceed to an uncoordinated implementation, which

- Automation allows to run more tests in less time and, thus, to run the tests more often leading to greater confidence in the system under test. Therefore, automation is a prerequisite for test-driven development, as developers run the tests for every bit of code they implement to successively grow the application.
- Also, the ability to quickly rerun an automated set of tests can help to shorten test execution time and to reduce the time-to-market when the bottleneck is repeating existing tests.

However, despite the potential efficiency gain that automated tests may provide, expectations about test automation are often unrealistically high. Test automation does not improve the effectiveness of testing (i.e. the total number of defects detected). Automating a test does not make it any more effective than running the same test manually. Usually, manual tests find even more defects than automated tests since it is most likely to find a defect the first time a test is run. If a test has been passed once, it is unlikely that a new defect will be detected when the same test is run again, unless the tested code is affected by a change. Furthermore, if testing is poorly organized, with ineffective tests that have a low capability of finding defects, automating these tests does not provide any benefits. Rather, the automation of a chaotic testing process only results in more and faster chaos.

Test automation is a significant investment. Although tools provide convenient features to automate testing, there is still a considerable amount of effort involved in planning, preparing, performing, and reporting on automated tests. And there is still a considerable amount of overhead involved in running the tests, including the deployment of the tests, the verification of the results, the handling of false alarms, and the maintenance of the test execution infrastructure. Automated tests have to be maintained too, as tests become obsolete or break because of changes that concern the user interface, output formats, APIs or protocols. In addition, the total cost of ownership of test tools involves not only the license fees but also additional costs such as training or dealing with technical problems since test tools are typically large and complex products.

The costs usually exceed the potential savings from faster and cheaper (automated) test execution. Thus, while it is sometimes argued that test automation pays off due to the reduced test execution cycles, in Web application testing the main benefit of automation comes from the advantages listed above that lead to improved quality and shorter time-to-market cycles. Even if the costs incurred by test automation may be higher compared with manual testing, the resulting benefits in quality and time call for this investment.

Thus, a sensible investment strategy uses tools to enhance manual testing, but does not aim to replace manual testing with automated testing. Manual tests are best to explore new functionality, driven by creativity, understanding, experience, and the gut feelings of a human tester. Automated tests secure existing functionality, find side-effects and defects that have been re-introduced, and enhance the range and accuracy of manual tests. Therefore, not all testing has to be automated. Partial automation can be very useful and various test tools are available to support the different kinds of testing activities.

7.7.2 Test Tools

Commonly used test tools support the following tasks:

- *Test planning and management*: These tools facilitate the management of test cases and test data, the selection of suitable test cases, and the collection of test results and bug tracking.

- *Test case design*: Tools available to design test cases support the developer in deriving test cases from the requirements definition or in generating test data.
- *Static and dynamic analyses*: Tools available to analyze Web applications, e.g., HTML validators or link checkers, try to discover deviations from standards.
- *Automating test runs*: Tools can automate test runs by simulating or logging as well as capturing and replaying the behavior of components or users.
- *System monitoring*: Tools available to monitor systems support us in detecting errors, e.g., by capturing system properties, such as memory consumption or database access.
- *General tasks*: Tools like editors or report generators are helpful and mentioned here for the sake of completeness. A detailed discussion would go beyond the scope of this book.

7.7.3 Selecting Test Tools

The current trend in test tools for Web applications is closely coupled with the continual evolution of Web technologies and modern development processes. A large number of different tools are available today. Descriptions of specific tools are normally of short validity, so they are omitted from this chapter. When selecting suitable tools for Web application testing, we always need to research and re-evaluate things. The test scheme introduced in this chapter can support us in selecting tools and building a well-structured and complete tool box. A comprehensive catalog of criteria for test tool evaluation is described in (Dustin et al. 2002). In addition, various Web pages maintain a continually updated summary of tools for Web application testing (e.g., at `http://www.softwareqatest.com/qatweb1.html`).

7.8 Outlook

The broad range of quality requirements on Web applications encourages a reorientation beyond the traditional focus on functions in testing and requires an approach to manage the effort of testing within the shrinking timeframes and budgets based on risk considerations. Users expect Web applications to be continually available, to have short response times, to be easy to use, to have a pleasant look and feel, to be highly secure, to be compatible with their Web browsers, to offer up-to-date data and, of course, to provide correctly implemented functionality. None of these quality characteristics in itself is new, and they are not new in the field of testing either. What is new is that, when developing Web applications, the combination of these quality characteristics has a decisive significance for the success of a Web application.

In traditional software development, some but not all quality characteristics are important, depending on the field of application (e.g., data-driven applications, real-time and embedded systems, or distributed applications). In contrast, when developing Web applications, it becomes increasingly important for the tester to have methods and tools suitable for each quality characteristic at his or her disposal.

The separation of a Web application's development from its operation is no longer part of evolutionary development approaches, which focus rather on continuous improvement and, consequently, introduce new testing challenges. Tests should be reusable across several development cycles, and they should be suitable for monitoring a Web application's operation. Since a Web application is subject to changes in the course of its development and operation, tests

should also be adaptable. This means that the significance of the quality requirements with regard to reusability and changeability increases for the tests themselves. The logical consequence is that automated testing becomes increasingly important for Web applications, the more so as a higher initial cost for test automation pays off by frequent test reuse especially in the maintenance phase of large-scale Web applications. Agile approaches, which build on short iterations and frequent delivery to eventually reach continuous integration, have already accepted test automation as an indispensable prerequisite.