In reality, all of the general characteristics of software quality discussed in Chapters 9, 15, and 26 apply to WebApps. However, the most relevant of these characteristics—usability, functionality, reliability, efficiency, and maintainability—provide a useful basis for assessing the quality of Web-based systems.

"If products are designed to better fit the natural tendencies of human behavior, then people will be more satisfied, more fulfilled, and more productive."

Susan Weinschenk

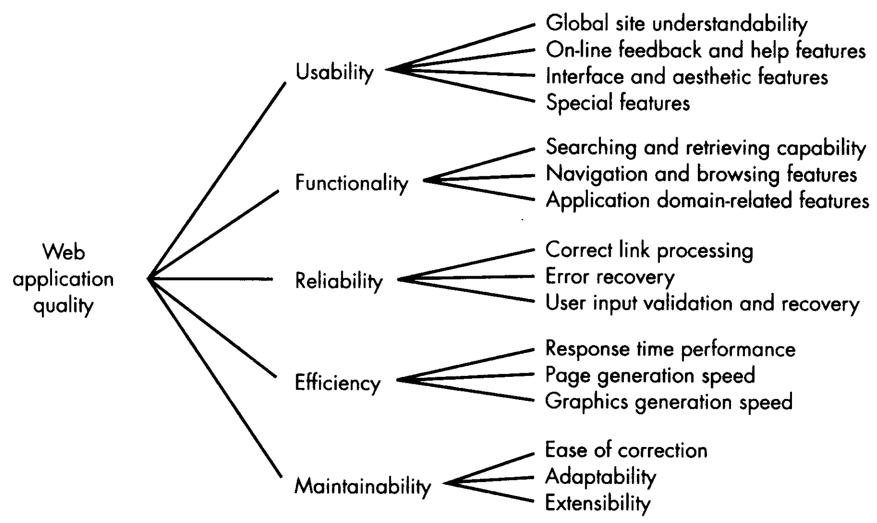
Olsina and his colleagues [OLS99] have prepared a "quality requirement tree" that identifies a set of technical attributes—usability, functionality, reliability, efficiency, and maintainability—that lead to high-quality WebApps.[1] Figure 19.1 summarizes their work. The criteria noted in the figure are of particular interest to Web engineers who must design, build, and maintain WebApps over the long term.

Offutt [OFF02] extends the five major quality attributes noted in Figure 19.1 by adding the following attributes:

What are the major attributes of quality for WebApps?

Security. WebApps have become heavily integrated with critical corporate and government databases. E-commerce applications extract and then store sensitive customer information. For these and many other reasons, WebApp security is paramount in many situations. The key measure of security is the ability of the WebApp and its server environment to rebuff unauthorized access and/or thwart an outright

FIGURE 19.1


Web application quality
Usability
Global site understandability
On-line feedback and help features
Interface and aesthetic features
Special features
Functionality
Searching and retrieving capability
Navigation and browsing features
Application domain-related features
Reliability
Correct link processing
Error recovery
User input validation and recovery
Efficiency
Response time performance
Page generation speed
Graphics generation speed
Maintainability
Ease of correction
Adaptability
Extensibility
Quality requirements tree [OLS99]

1 These quality attributes are quite similar to those presented in Chapters 9, 15, and 26. The implication: quality characteristics are universal for all software.

malevolent attack. A detailed discussion of WebApp security is beyond the scope of this book. The interested reader should see [MCC01], [NOR02], or [KAL03].

Availability. Even the best WebApp will not meet users' needs if it is unavailable. In a technical sense, availability is the measure of the percentage of time that a WebApp is available for use. The typical end-user expects WebApps to be available 24/7/365. Anything less is deemed unacceptable.[2] But "up-time" is not the only indicator of availability. Offutt [OFF02] suggests that "using features available on only one browser or one platform" makes the WebApp unavailable to those with a different browser/platform configuration. The user will invariably go elsewhere.

Scalability. Can the WebApp and its server environment be scaled to handle 100, 1000, 10,000, or 100,000 users? Will the WebApp and the systems with which it is interfaced handle significant variation in volume or will responsiveness drop dramatically (or cease altogether)? It is not enough to build a WebApp that is successful. It is equally important to build a WebApp that can accommodate the burden of success (significantly more end-users) and become even more successful.

Time-to-market. Although time to market is not a true quality attribute in the technical sense, it is a measure of quality from a business point of view. The first WebApp in the market often captures a disproportionate number of end-users.

INFO

WebApp Design Quality Checklist

The following checklist, adapted from information presented at Webreference.com, provides a set of questions that will help both Web-engineers and end-users assess overall WebApp quality:

- Can content and/or function and/or navigation options be tailored to the user's preferences?
- Can content and/or functionality be customized to the bandwidth over which the user communicates.
- Have graphics and other nontext media been used appropriately? Are graphics file sizes optimized for display efficiency?
- Are tables organized and sized in a manner that makes them understandable and displayed efficiently?
- Is HTML optimized to eliminate inefficiencies?
- Is the overall page design easy to read and navigate?
- Do all pointers (links) provide links to information that is of interest to users?
- Is it likely that most links have persistence on the Web?
- Is the WebApp instrumented with site management utilities that include tools for usage tracking, link testing, local searching, and security?

Billions of Web pages are available for those in search of information on the World Wide Web. Even well-targeted Web searches result in an avalanche of content. With so many sources of information to choose from, how does the user assess the quality (e.g., veracity, accuracy, completeness, timeliness) of the content that is presented within a WebApp? Tillman [TIL00] suggests a useful set of criteria for assessing the quality of content:

2 This expectation is, of course, unrealistic. Major WebApps must schedule "downtime" for fixes and upgrades.

What should we consider when assessing content quality?

- Can the scope and depth of content be easily determined to ensure that it meets the user's needs?
- Can the background and authority of the content's authors be easily identified?
- Is it possible to determine the currency of the content, the last update, and what was updated?
- Is the content and its location stable (i.e., will it remain at the referenced URL)?

In addition these content-related questions, the following might be added:

- Is content credible?
- Is content unique? That is, does the WebApp provide some unique benefit to those who use it?
- Is content valuable to the targeted user community?
- Is content well-organized? Indexed? Easily accessible?

The checklists noted in this section represent only a small sampling of the issues that should be addressed as the design of a WebApp evolves. An important goal of Web engineering is to develop systems in which affirmative answers are provided to all quality-related questions.

"Just because you can, doesn't mean you should."

Jean Kaiser

19.1.2 Design Goals

In her regular column on Web design, Jean Kaiser [KAI02] suggests the following design goals that are applicable to virtually every WebApp regardless of application domain, size, or complexity:

Simplicity. Although it may seem old-fashioned, the aphorism "all things in moderation" applies to WebApps. There is a tendency among some designers to provide the end-user with "too much"—exhaustive content, extreme visuals, intrusive animation, enormous Web pages, the list is long. Better to strive for moderation and simplicity.

Consistency. This design goal applies to virtually every element of the design model. Content should be constructed consistently (e.g., text formatting and font styles should be the same across all text documents; graphic art should have a consistent look, color scheme, and style). Graphic design (aesthetics) should present a consistent look across all parts of the WebApp. Architectural design should establish templates that lead to a consistent hypermedia structure. Interface design should define consistent modes of interaction, navigation, and content display. Navigation mechanisms should be used consistently across all WebApp elements.

Identity. The aesthetic, interface, and navigational design of a WebApp must be consistent with the application domain for which it is to be built. A Web site for a hip-hop and rewarding experience. Interface design concepts, principles, and methods provide the Web engineer with the tools required to achieve this list of attributes.

In Chapter 12, we noted that interface design begins not with a consideration of technology, but with a careful examination of the end-user. During analysis modeling for Web engineering (Chapter 18), a user hierarchy is developed. Each user category may have subtly different needs, may want to interact with the WebApp in different ways, and may require unique functionality and content. This information is derived during requirements analysis, but it is revisited as the first step in interface design.

"If a site is perfectly usable but it lacks an elegant and appropriate design style, it will fail."

Curt Cloninger

ADVICE

If it is likely that users may enter your WebApp at various locations and levels in the content hierarchy, be sure to design every page with navigation features that will lead the user to other points of interest.

Dix [DIX99] argues that a Web engineer must design an interface so that it answers three primary questions for the end-user:

Where am I? The interface should (1) provide an indication of the WebApp that has been accessed[4] and (2) inform the user of her location in the content hierarchy.

What can I do now? The interface should always help the user understand his current options—what functions are available, what links are live, what content is relevant.

Where have I been; where am I going? The interface must facilitate navigation. Hence, it must provide a "map" (implemented in a way that is easy to understand) of where the user has been and what paths may be taken to move elsewhere within the WebApp.

An effective WebApp interface must provide answers for each of these questions as the end-user navigates through content and functionality.

19.3.1 Interface Design Principles and Guidelines

Bruce Tognozzi [TOG01] defines a set of fundamental characteristics that all interfaces should exhibit and, in doing so, establishes a philosophy that should be followed by every WebApp interface designer:

KEY POINT

A good WebApp interface is understandable and forgiving, providing the user with a sense of control.

> Effective interfaces are visually apparent and forgiving, instilling in their users a sense of control. Users quickly see the breadth of their options, grasp how to achieve their goals, and do their work.
>
> Effective interfaces do not concern the user with the inner workings of the system. Work is carefully and continuously saved, with full option for the user to undo any activity at any time.
>
> Effective applications and services perform a maximum of work, while requiring a minimum of information from users.

4 Each of us has bookmarked a Web-site page, only to revisit later and have no indication of the Web site or the context for the page (as well as no way to move to another location within the site).

In order to design interfaces that exhibit these characteristics, Tognozzi [TOG01] identifies a set of overriding design principles:[5]

Anticipation—A WebApp should be designed so that it anticipates the user's next move. For example, consider a customer support WebApp developed by a manufacturer of computer printers. A user has requested a content object that presents information about a printer driver for a newly released operating system. The designer of the WebApp should anticipate that the user might request a download of the driver and should provide navigation facilities that allow this to happen without requiring the user to search for this capability.

KEY POINT
A WebApp interface should be designed to conform to the set of principles noted here.

Communication—The interface should communicate the status of any activity initiated by the user. Communication can be obvious (e.g., a text message) or subtle (e.g., a sheet of paper moving through a printer to indicate that printing is underway). The interface should also communicate user status (e.g., the user's identification) and location within the WebApp content hierarchy.

Consistency—The use of navigation controls, menus, icons, and aesthetics (e.g., color, shape, layout) should be consistent throughout the WebApp. For example, if underlined blue text implies a navigation link, content should never incorporate blue underlined text that does not imply a link. Every feature of the interface should respond in a manner that is consistent with user expectations.[6]

Controlled autonomy—The interface should facilitate user movement throughout the WebApp, but it should do so in a manner that enforces navigation conventions that have been established for the application. For example, navigation to secure portions of the WebApp should be controlled by userID and password, and there should be no navigation mechanism that enables a user to circumvent these controls.

Efficiency—The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the Web engineer who designs and builds it or the client-server environment that executes it. Tognozzi [TOG01] discusses this when he writes: "This simple truth is why it is so important for everyone involved in a software project to appreciate the importance of making user productivity goal one and to understand the vital difference between building an efficient system and empowering an efficient user."

Flexibility—The interface should be flexible enough to enable some users to accomplish tasks directly and others to explore the WebApp in a somewhat random fashion. In every case, it should enable the user to understand where he is and provide the user with functionality that can undo mistakes and retrace poorly chosen navigation paths.

5 Tognozzi's original principles have been adapted and extended for use in this book. See [TOG01] for further discussion of these principles.

6 Tognozzi [TOG01] notes that the only way to be sure that user expectations are properly understood is through comprehensive user testing (Chapter 20).

Focus—The WebApp interface (and the content it presents) should stay focused on the user task(s) at hand. In all hypermedia there is a tendency to route the user to loosely related content. Why? Because it's very easy to do! The problem is that the user can rapidly become lost in many layers of supporting information and lose site of the original content that she wanted in the first place.

Fitt's Law—"The time to acquire a target is a function of the distance to and size of the target" [TOG01]. Based on a study conducted in the 1950s [FIT54], Fitt's Law "is an effective method of modeling rapid, aimed movements, where one appendage (like a hand) starts at rest at a specific start position, and moves to rest within a target area" [ZHA02]. If a sequence of selections or standardized inputs (with many different options within the sequence) is defined by a user task, the first selection (e.g., mouse pick) should be physically close to the next selection. For example, consider a WebApp home page interface at an e-commerce site that sells consumer electronics.

Each user option implies a set of follow-on user choices or actions. For example, a "buy a product" option requires that the user enter a product category followed by the product name. The product category (e.g., audio equipment, televisions, DVD players) appears as a pull-down menu as soon as "buy a product" is picked. Therefore, the next choice is immediately obvious (it is nearby), and the time to acquire it is negligible. If, on the other hand, the choice appeared on a menu that was located on the other side of the screen, the time for the user to acquire it (and then make the choice) would be far too long.

WebRef
A search on the Web will uncover many available libraries, e.g., Java API packages, interfaces, and classes at java.sun.com or COM, DCOM, and Type Libraries at msdn.Microsoft.com.

Human interface objects—A vast library of reusable human interface objects has been developed for WebApps. Use them. Any interface object that can be "seen, heard, touched or otherwise perceived" [TOG01] by an end-user can be acquired from any one of a number of object libraries.

Latency reduction—Rather than making the user wait for some internal operation to complete (e.g., downloading a complex graphical image), the WebApp should use multitasking in a way that lets the user proceed with work as if the operation has been completed. In addition to reducing latency, delays must be acknowledged so that the user understands what is happening. This includes (1) providing audio feedback (e.g., a click or bell tone) when a selection does not result in an immediate action by the WebApp; (2) displaying an animated clock or progress bar to indicate that processing is under way; (3) provide some entertainment (e.g., an animation or text presentation) while lengthy processing occurs.

"The best journey is the one with the fewest steps. Shorten the distance between the user and their goal."
Author unknown

Learnability—A WebApp interface should be designed to minimize learning time, and once learned, to minimize relearning required when the WebApp is revisited. In general the interface should emphasize a simple, intuitive design that organizes content and functionality into categories that are obvious to the user.

Metaphors—An interface that uses an interaction metaphor is easier to learn and easier to use, as long as the metaphor is appropriate for the application and the user. A metaphor should call on images and concepts from the user's experience, but it does not need to be an exact reproduction of a real world experience. For example, an e-commerce site that implements automated bill paying for a financial institution uses a checkbook metaphor (not surprisingly) to assist the user in specifying and scheduling bill payments. However, when a user "writes" a check, he need not enter the complete payee name but can pick from a list of payees or have the system select based on the first few typed letters. The metaphor remains intact, but the user gets an assist from the WebApp.

ADVICE

Metaphors are an excellent idea because they mirror real world experience. Just be sure that the metaphor you choose is well known among end-users.

Maintain work product integrity. A work product (e.g., a form completed by the user, a user specified list) must be automatically saved so that it will not be lost if an error occurs. Each of us has experienced the frustration associated with completing a lengthy WebApp form only to have the content lost because of an error (made by us, by the WebApp, or in transmission from client to server). To avoid this, a WebApp should be designed to auto-save all user specified data.

Readability—All information presented through the interface should be readable by young and old. The interface designer should emphasize readable type styles, font sizes, and color background choices that enhance contrast.

Track state—When appropriate, the state of the user interaction should be tracked and stored so that a user can log-off and return later to pick up where she left off. In general, cookies can be designed to store state information. However, cookies are a controversial technology, and other design solutions may be more palatable for some users.

Visible navigation—A well-designed WebApp interface provides "the illusion that users are in the same place, with the work brought to them" [TOG01]. When this approach is used, navigation is not a user concern. Rather, the user retrieves content objects and selects functions that are displayed and executed through the interface.

SAFEHOME

Interface Design Review

The scene: Doug Miller's office.

The players: Doug Miller (manager of the *SafeHome* software engineering group) and Vinod Raman, a member of the *SafeHome* product software engineering team.

The conversation:

Doug: Vinod, have you and the team had a chance to review the SafeHomeAssured.com e-commerce interface prototype?

Vinod: Yeah . . . we all went through it from a technical point of view, and I have a bunch of notes. I e-mailed 'em to Sharon [manager of the Web engineering team for the outsourcing vendor for the *SafeHome* e-commerce Web site] yesterday.

Doug: You and Sharon can get together and discuss the small stuff . . . give me a summary of the important issues.

Vinod: Overall, they've done a good job, nothing ground breaking, but it's a typical e-commerce interface, decent aesthetics, reasonable layout. They've hit all the important functions. . . .

Doug (smiling ruefully): But?

Vinod: Well, there are a few things. . . .

Doug: Such as . . . ?

Vinod (showing Doug a sequence of storyboards for the interface prototype): Here's the major functions menu that's displayed on the home page:

Learn about *SafeHome*
Describe your home
Get *SafeHome* component recommendations
Purchase a *SafeHome* system
Get technical support

The problem isn't with these functions, they're all okay, but the level of abstraction isn't right.

Doug: They're all major functions, aren't they?

Vinod: They are, but here's the thing . . . you can purchase a system by inputting a list of components . . . no real need to describe the house, if you don't want to. I'd suggest only four menu options on the home page:

Learn about *SafeHome*
Specify the *SafeHome* system you need
Purchase a *SafeHome* system
Get technical support

When you select **specify the *SafeHome* system you need,** you'll then have the following options:

Select *SafeHome* components
Get *SafeHome* component recommendations

If you're a knowledgeable user, you'll select components from a set of categorized pull-down menus for sensors, cameras, control panels, etc. If you need help, you'll ask for a recommendation and that will require that you describe your house. I think it's a bit more logical.

Doug: I agree. Have you talked with Sharon about this?

Vinod: No, I want to discuss this with marketing first, and then I'll give her a call.

Nielsen and Wagner [NIE96] suggest a few pragmatic interface design guidelines (based on their redesign of a major WebApp) that provide a nice complement to the principles suggested earlier in this section:

- Reading speed on a computer monitor is approximately 25 percent slower than reading speed for hardcopy. Therefore, do not force the user to read voluminous amounts of text, particularly when the text explains the operation of the WebApp or assists in navigation.
- Avoid "under construction" signs—they raise expectations and cause an unnecessary link that is sure to disappoint.
- Users prefer not to scroll. Important information should be placed within the dimensions of a typical browser window.
- Navigation menus and head bars should be designed consistently and should be available on all pages that are available to the user. The design should not rely on browser functions to assist in navigation.
- Aesthetics should never supersede functionality. For example, a simple button might be a better navigation option than an aesthetically pleasing, but vague image or icon whose intent is unclear.