19.12. Use UML to develop three or four design representations for content objects that would be encountered as the "learning engine" described in Problem 19.2 is designed.

19.13. Do a bit of additional research on the MVC architecture and decide whether it would be an appropriate WebApp architecture for the "learning engine" discussed in Problem 19.2.

19.14. What is the difference between navigation syntax and navigation semantics?

19.15. Do some research and present two or three complete hypermedia design patterns to your class.

19.16. Define two or three NSUs for the SafeHomeAssured.com WebApp. Describe each in some detail.

FURTHER READINGS AND INFORMATION SOURCES

Although hundreds of books have been written on "Web design," very few discuss any meaningful technical methods for doing design work. At best, a variety of useful guidelines for WebApp design are presented, worthwhile examples of Web pages and Java programming are shown, and the technical details important for implementing modern WebApps are discussed. Among the many offerings in this category, Powell's encyclopedic discussion [POW00] is worth considering. In addition, books by Galitz [GAL02], Heinicke [HEI02], Schmitt (*Designing CSS Web Pages,* New Riders Publishing, 2002), Donnelly (*Designing Easy-to-Use Websites,* Addison-Wesley, 2001), and Nielsen [NIE00] provide much useful guidance.

The agile view of design (and other topics) for WebApps is presented by Wallace and his colleagues (*Extreme Programming for Web Projects,* Addison-Wesley, 2003). Conallen (*Building Web Applications with UML,* second edition, Addison-Wesley, 2002) and Rosenberg and Scott (*Applying Use-Case Driven Object Modeling with UML,* Addison-Wesley, 2001) present detailed examples of WebApps modeled using UML.

Van Duyne and his colleagues (*The Design of Sites: Patterns, Principles and Processes,* Addison-Wesley, 2002) have written an excellent book that covers most important aspects of the Web engineering design process. Design process models and design patterns are covered in detail. Wodtke (*Information Architecture,* New Riders Publishing, 2003), Rosenfeld and Morville (*Information Architecture for the World Wide Web,* O'Reilly & Associates, 2002), and Reiss (*Practical Information Architecture,* Addison-Wesley, 2000) address content architecture and other topics.

Design techniques are also mentioned in books written about specific development environments. Interested readers should examine books on J2EE, Java, ASP.NET, CSS, XML, Perl, and a variety of WebApp creation applications (*Dreamweaver, HomePage, Frontpage, GoLive, MacroMedia Flash,* etc.) for useful design tidbits.

A wide variety of information sources on design for Web engineering is available on the Internet. An up-to-date list of World Wide Web references can be found at the SEPA Web site: **http://www.mhhe.com/pressman.**

CHAPTER

20 TESTING FOR WEBAPPS

KEY CONCEPTS

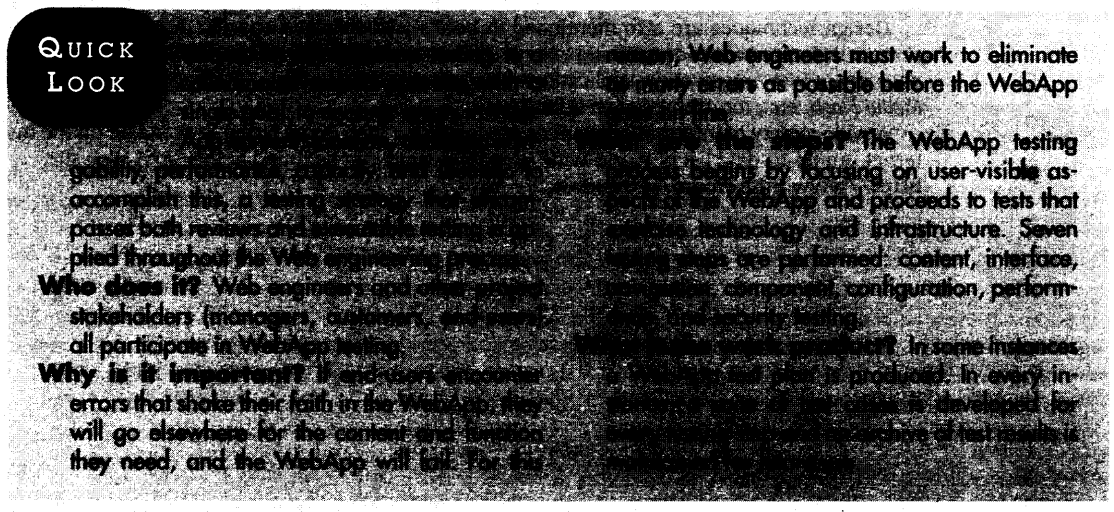
error characteristics
quality dimensions
strategy
testing
- compatibility
- component-level
- configuration
- content
- database
- load
- performance
- navigation
- stress
- usability
- user interface

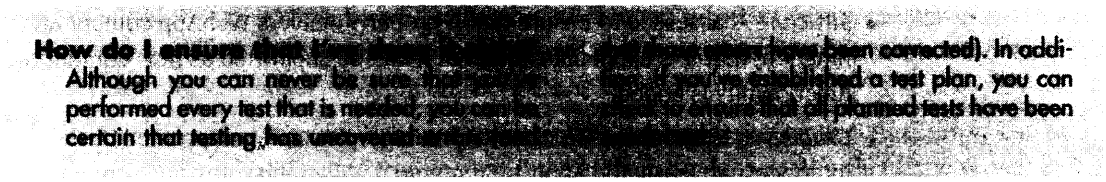
There is an urgency that always pervades the Web engineering process. As formulation, planning, analysis, design, and construction are conducted, stakeholders—concerned about competition from other WebApps, coerced by customer demands, and worried that they'll miss a market window—press to get the WebApp on-line. As a consequence, technical activities that often occur late in the Web engineering process, such as WebApp testing, are sometimes given short shrift. This can be a catastrophic mistake. To avoid it, the Web engineering team must ensure that each WebE work product exhibits high quality. Wallace and his colleagues [WAL03] note this when they state:

> Testing shouldn't wait until the project is finished. Start testing before you write one line of code. Test constantly and effectively, and you will develop a much more durable Web site.

Since analysis and design models cannot be tested in the classical sense, the Web engineering team should conduct formal technical reviews (Chapter 26) as well as executable tests. The intent is to uncover and correct errors before the WebApp is made available to its end-users.

QUICK LOOK





20.1 Testing Concepts for WebApps

In Chapter 13, we noted that testing is the process of exercising software with the intent of finding (and ultimately correcting) errors. This fundamental philosophy does not change for WebApps. In fact, because Web-based systems and applications reside on a network and interoperate with many different operating systems, browsers (or other interface devices such as PDAs or mobile phones), hardware platforms, communications protocols, and "backroom" applications, the search for errors represents a significant challenge for Web engineers.

To understand the objectives of testing within a Web engineering context, we must consider the many dimensions of WebApp quality.[1] In the context of this discussion, we consider quality dimensions that are particularly relevant in any discussion of testing for Web engineering work. We also consider the nature of the errors that are encountered as a consequence of testing, and the testing strategy that is applied to uncover these errors.

20.1.1 Dimensions of Quality

Quality is incorporated into a Web application as a consequence of good design. It is evaluated by applying a series of technical reviews that assess various elements of the design model and by applying a testing process that is discussed throughout this chapter. Both reviews and testing examine one or more of the following quality dimensions [MIL00]:

How do we assess quality within the context of a WebApp and its environment?

- *Content* is evaluated at both a syntactic and semantic level. At the syntactic level, spelling, punctuation, and grammar are assessed for text-based documents. At a semantic level, correctness (of information presented), consistency (across the entire content object and related objects), and lack of ambiguity are all assessed.
- *Function* is tested to uncover errors that indicate lack of conformance to customer requirements. Each WebApp function is assessed for correctness, instability, and general conformance to appropriate implementation standards (e.g., Java or XML language standards).

1 WebApp quality has also been considered in Chapter 19.

Because many WebApps evolve continuously, WebApp testing is an on-going activity conducted by Web support staff who use regression tests derived from the tests developed when the WebApp was first engineered.

20.1.4 Test Planning

The use of the word *planning* (in any context) is anathema to some Web developers. As we noted in earlier chapters, these developers just start—hoping that a killer WebApp will emerge. A Web engineer recognizes that planning establishes a roadmap for all work that follows. It's worth the effort.

In their book on WebApp testing, Splaine and Jaskiel [SPL01] state:

> Except for the simplest of Web sites, it quickly becomes apparent that some sort of test planning is needed. All too often, the initial number of bugs found from ad hoc testing is large enough that not all of them are fixed the first time they're detected. This puts an additional burden on people who test Web sites and applications. Not only must they conjure up imaginative new tests, but they must also remember how previous tests were executed in order to reliably re-test the Web site/application, and ensure that known bugs have been removed and that no new bugs have been introduced.

The test plan identifies a testing task set, the work products to be developed, and the way in which results are to be evaluated, recorded, and reused.

The question for every Web engineer is: How do we "conjure up imaginative new tests," and what should those tests focus on? The answers to these questions are contained within a test plan.

A WebApp test plan identifies (1) a task set[2] to be applied as testing commences, (2) the work products to be produced as each testing task is executed, and (3) the manner in which the results of testing are evaluated, recorded, and reused when regression testing is conducted. In some cases, the test plan is integrated with the project plan. In others, the test plan is a separate document.

20.2 The Testing Process—An Overview

The testing process for Web engineering begins with tests that exercise content and interface functionality that is immediately visible to end-users. As testing proceeds, aspects of the design architecture and navigation are exercised. The user may or may not be cognizant of these WebApp elements. Finally, the focus shifts to tests that exercise technological capabilities that are not always apparent to end-users—WebApp infrastructure and installation/implementation issues.

> **"In general, the software testing techniques [Chapters 13 and 14] that are applied to other applications are the same as those applied to Web-based applications . . . The difference between the two types of testing is that the technology variables in the Web environment multiply."**
>
> **Hung Nguyen**

2 Task sets are discussed in Chapter 2. A related term--*work flow*—has also been used in this book to describe a series of tasks required to accomplish a software engineering activity.

FIGURE 20.1
The testing process

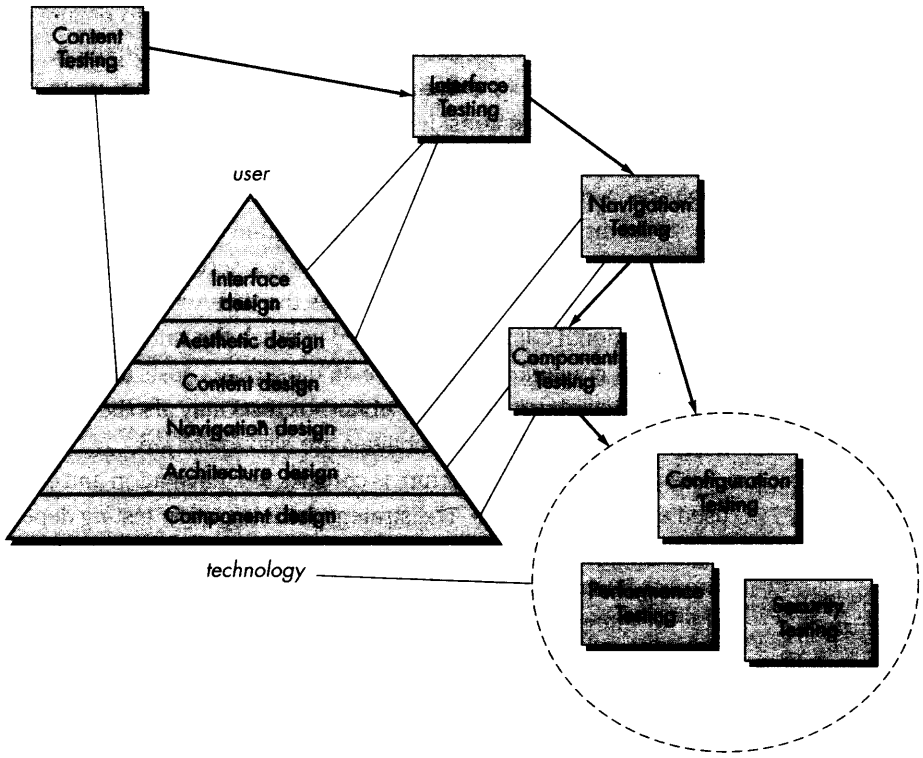


Figure 20.1 juxtaposes the WebApp testing process with the design pyramid discussed in Chapter 19. Note that as the testing flow proceeds from left to right and top to bottom, user visible elements of the WebApp design (top elements of the pyramid) are tested first, followed by infrastructure design elements.

Content testing (and reviews) attempts to uncover errors in content. This testing activity is similar in many respects to copy-editing for a written document. In fact, a large Web site might enlist the services of a professional copy editor to uncover typographical errors, grammatical mistakes, errors in content consistency, errors in graphical representations, and cross referencing errors. In addition to examining static content for errors, this testing step also considers dynamic content derived from data maintained as part of a database system that has been integrated with the WebApp.

Interface testing exercises interaction mechanisms and validates aesthetic aspects of the user interface. The intent is to uncover errors that result from poorly implemented interaction mechanisms or omissions, inconsistencies or ambiguities that have been introduced into the interface inadvertently.

Navigation testing applies use-cases, derived as part of the analysis activity, in the design of test cases that exercise each usage scenario against the navigation design.

Navigation mechanisms (e.g., menu bars) implemented within the interface layout are tested against use-cases and NSUs (Chapter 19) to ensure that any errors that impede completion of a use-case are identified and corrected.

Component testing exercises content and functional units within the WebApp. When WebApps are considered, the concept of the unit (introduced in Chapter 13) changes. The "unit" of choice within the content architecture (Chapter 19) is the Web page. Each Web page encapsulates content, navigation links, and processing elements (forms, scripts, applets). A "unit" within the WebApp architecture may be a defined functional component that provides service directly to an end-user or an infrastructure component that enables the WebApp to perform all of its capabilities. Each functional component is tested in much the same way as an individual module is tested in conventional software. In most cases, tests are black-box oriented. However, if processing is complex, white-box tests may also be used.[3] In addition to functional testing, database capabilities are also exercised.

The strategy for integration testing depends upon the WebApp architecture that has been chosen during design.

As the WebApp architecture is constructed, navigation and component testing are used as *integration tests*. The strategy for integration testing depends on the content and WebApp architecture that has been chosen (Chapter 19). If the content architecture has been designed with a linear, grid, or simple hierarchical structure, it is possible to integrate Web pages in much the same way as we integrate modules for conventional software. However, if a mixed hierarchy or network (Web) architecture is used, integration testing is similar to the approach used for OO systems. Thread-based testing (Chapter 14) can be used to integrate the set of Web pages (a NSU may be used to define the appropriate set) required to respond to a user event. Each thread is integrated and tested individually. Regression testing is applied to ensure that no side effects occur. Cluster testing integrates a set of collaborating pages (determined by examining the use-cases and NSU). Test cases are derived to uncover errors in the collaborations.

Each element of the WebApp architecture is unit tested to the extent possible. For example, in a MVC architecture (Chapter 19) the *model*, *view* and *controller* components are each tested individually. Upon integration, the flow of control and data across each of these elements is assessed in detail.

Configuration testing attempts to uncover errors that are specific to a particular client or server environment. A cross-reference matrix that defines all probable operating systems, browsers,[4] hardware platforms, and communications protocols is created. Tests are then conducted to uncover errors associated with each possible configuration.

---

3 Black-box and white-box testing techniques are discussed in Chapter 14.

4 Browsers are notorious for implementing their own subtly different "standard" interpretations of HTML and Javascript.