**QUICK LOOK**

Design focuses on integration of existing components and the creation of new components. Implementation and testing strive to exercise both client and server functionality within the context of component integration standards and the architecture.

**What is the work product?** A high-quality client/server system is the outcome of c/s software engineering. Other software work products (discussed earlier in this book) are also produced.

**How do I ensure that I've done it right?** Use the same SQA practices that are applied in every software engineering process—formal technical reviews assess the analysis and design models, specialized reviews consider issues associated with component integration and middleware, and testing is applied to uncover errors at the component, subsystem, client, and server levels.

In this chapter, we examine a dominant architecture for information processing—*client/server* (c/s) systems. Client/server systems have evolved in conjunction with advances in desktop computing, component-based software engineering, new storage technologies, improved network communications, and enhanced database technology. The objective of this chapter[1] is to present a brief overview of client/server systems with an emphasis on the special software engineering issues that must be addressed when such c/s systems are analyzed, designed, tested, and supported.

## 28.1 THE STRUCTURE OF CLIENT/SERVER SYSTEMS

**Quote:**

"The c/s computing model represents a specific instance of distributed cooperative processing, where the relationship between clients and servers is the relationship of both hardware and software components."

**Alex Berson**

Hardware, software, database, and network technologies all contribute to distributed and cooperative computer architectures. In its most general form, a distributed and cooperative computer architecture is as illustrated in Figure 28.1. A *root system,* sometimes a mainframe, serves as the repository for corporate data. The root system is connected to servers (typically powerful workstations or PCs) that play a dual role. The servers update and request corporate data maintained by the root system. They also maintain local departmental systems and play a key role in networking user-level PCs via a local area network (LAN).

In a c/s structure, the computer that resides above another computer (in Figure 28.1) is called the *server,* and the computer(s) at the level below is called the *client.* The client requests services,[2] and the server provides them. However, within the context of the the architecture represented in Figure 28.2, a number of different implementations can be achieved [ORF99]:

1 Portions of this chapter have been adapted from course material developed by John Porter for the client/server curriculum offered at The BEI Engineering School of Fairfield University. Used with permission.

2 In this context, *services* can be broadly interpreted to mean data, processing, or a combination of the two.

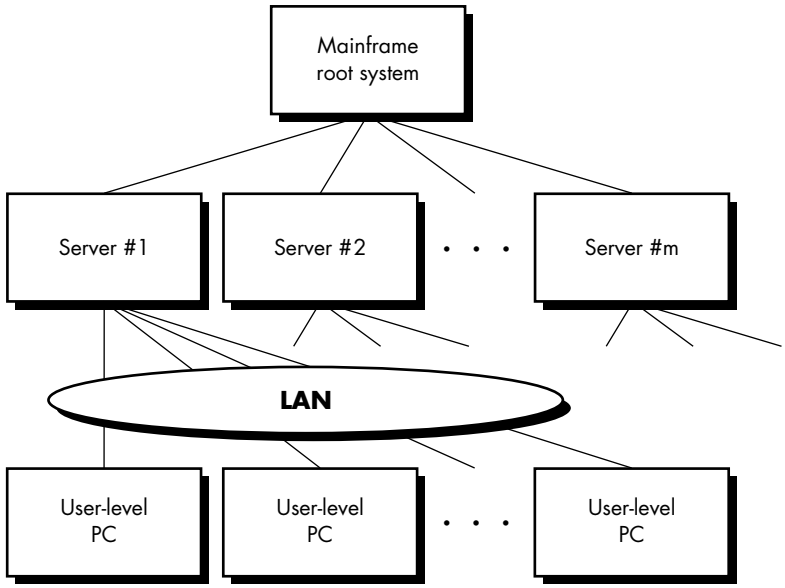



**FIGURE 28.1**
Distributed, cooperative computer architectures in a corporate setting

**File servers.** The client requests specific records from a file. The server transmits these records to the client across the network.

**Database servers.** The client sends *structured query language* (SQL) requests to the server. These are transmitted as messages across the network. The server processes the SQL request and finds the requested information, passing back the results only to the client.

**Transaction servers.** The client sends a request that invokes remote procedures at the server site. The remote procedures are a set of SQL statements. A *transaction* occurs when a request results in the execution of the remote procedure with the result transmitted back to the client.

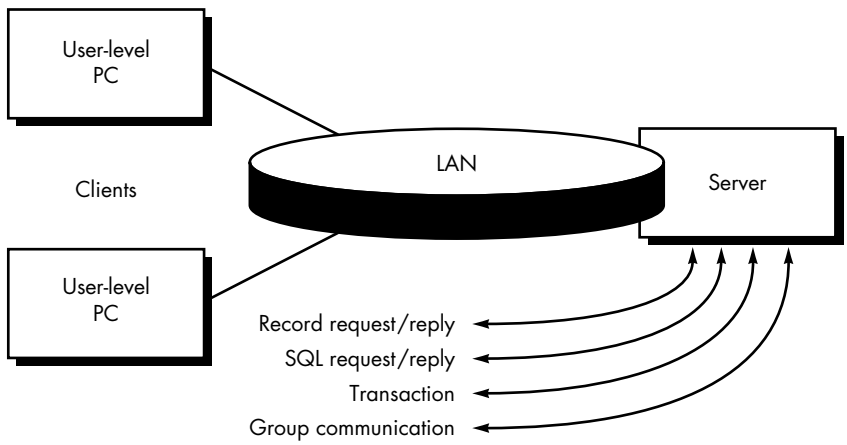



**FIGURE 28.2**
Client/server options

**Groupware servers.** When the server provides a set of applications that enable communication among clients (and the people using them) using text, images, bulletin boards, video, and other representations, a groupware architecture exists.

### 28.1.1 Software Components for c/s Systems

Instead of viewing software as a monolithic application to be implemented on one machine, the software that is appropriate for a c/s architecture has several distinct subsystems that can be allocated to the client, the server, or distributed between both machines:


The FAQ for the **comp.client-server** newsgroup can be found at **www.faqs.org/faqs/client-server-faq/**

**User interaction/presentation subsystem.** This subsystem implements all functions that are typically associated with a graphical user interface.

**Application subsystem.** This subsystem implements the requirements defined by the application within the context of the domain in which the application operates. For example, a business application might produce a variety of printed reports based on numeric input, calculations, database information, and other considerations. A groupware application might provide the facilities for enabling bulletin board communication or e-mail. In both cases, the application software may be partitioned so that some components reside on the client and others reside on the server.

**Database management subsystem.** This subsystem performs the data manipulation and management required by an application. Data manipulation and management may be as simple as the transfer of a record or as complex as the processing of sophisticated SQL transactions.


Middleware establishes the infrastructure that enables c/s software components to interoperate.

In addition to these subsystems, another software building block, often called *middleware*, exists in all c/s systems. Middleware comprises software components that exist on both the client and the server and includes elements of network operating systems as well as specialized application software that supports database-specific applications, object request broker standards (Section 28.1.5), groupware technologies, communication management, and other features that facilitate the client/server connection. Orfali, Harkey, and Edwards [ORF99] have referred to middleware as "the nervous system of a client/server system."

### 28.1.2 The Distribution of Software Components

Once the basic requirements for a client/server application have been determined, the software engineer must decide how to distribute the software components that constitute the subsystems discussed in Section 28.1.1 between the client and the server. When most of the functionality associated with each of the three subsystems is allocated to the server, a *fat server* design has been created. Conversely, when the

A "fat" client implements most application-specific functions at the client. A "thin" client relegates most processing to the server.

client implements most of the user interaction/presentation, application, and database components, a *fat client* design has been created.

Fat clients are commonly encountered when file server and database server architectures are implemented. In this case, the server provides data management support, but all application and GUI software resides at the client. Fat servers are often designed when transaction and groupware systems are implemented. The server provides application support required to respond to transactions and communication from the clients. The client software focuses on GUI and communication management.

Fat clients and fat servers can be used to illustrate the general approach for the allocation of client/server software systems. However, a more granular approach to software component allocation defines five different configurations:

**What are configuration options for c/s software components?**

**Distributed presentation.** In this rudimentary client/server approach, database logic and the application logic remain on the server, typically a mainframe. The server also contains the logic for preparing screen information, using software such as CICS. Special PC-based software is used to convert character-based screen information transmitted from the server into a GUI presentation on a PC.

**Remote presentation.** An extension of the distributed presentation approach, primary database and application logic remain on the server, and data sent by the server is used by the client to prepare the user presentation.

**Distributed logic.** The client is assigned all user presentation tasks and the processes associated with data entry, such as field-level validation, server query formulation, and server update information and requests. The server is assigned database management tasks and the processes for client queries, server file updates, client version control, and enterprise-wide applications.

**Remote data management.** Applications on the server create a new data source by formatting data that have been extracted from elsewhere (e.g., from a corporate level source). Applications allocated to the client are used to exploit the new data that has been formatted by the server. Decision support systems are included in this category.

**Distributed databases.** The data forming the database is spread across multiple servers and clients. Therefore, the client must support data management software components as well as application and GUI components.

In recent years, there has also been considerable emphasis on thin-client technology. A *thin client* is a so-called "network computer" that relegates all application processing to a fat server. Thin clients (network computers) offer substantially lower per unit cost at little or no significant performance loss when compared to desktop machines.

### 28.1.3 Guidelines for Distributing Application Subsystems

While no absolute rules cover the distribution of application subsystems between the client and server, the following guidelines are generally followed:

**The presentation/interaction subsystem is generally placed on the client.** The availability of PC-based, Windows-based environments and the computing power required for a graphical user interface makes this approach cost effective.

**If the database is to be shared by multiple users connected by the LAN, it is typically located on the server.** The database management system and the database access capability are also located on the server together with the physical database.

**Static data that are used for reference should be allocated to the client.** This places the data closest to the users that require them and minimizes unnecessary network traffic and loading on the server.

**Quote:**

"Some analysts view client/server computing as the fourth wave of [change in the history of] computing."

**Bernard Boar**

The balance of the application subsystem is distributed between the client and server based on the distribution that optimizes the server and client configurations and the network that connects them. For example, the implementation of a mutually exclusive relationship typically involves a search of the database to determine if there is a record that matches the parameters for a search pattern. If no match is found, an alternate search pattern is used. If the application that controls this search pattern is contained fully on the server, network traffic is minimized. The first network transmission from the client to the server would contain the parameters for both the primary and secondary search patterns. Application logic on the server would determine if the secondary search is required. The response message to the client would contain the record found as a result of either the primary or the secondary search. The alternate approach of placing on the client the logic to determine if a second search is required would involve a message for the first record retrieval, a response over the network if the record is not found, a second message containing the parameters for the second search, and a final response with the retrieved record. If the second search is required 50 percent of the time, placing the logic on the server to evaluate the first search and initiate the second search, if necessary, would reduce network traffic by 33 percent.

**ADVICE**

*Although distribution guidelines are worthwhile, every system must be considered on its own merits. For every benefit derived from, say, a fat client, the designer must contend with an equal set of negatives.*

The final decision on subsystem distribution should be based not only on the individual application but on the mix of applications operating on the system. For example, an installation might contain some applications that require extensive GUI processing and little central database processing. This would lead to the use of powerful workstations on the client side, and a bare bones server. With this configuration in place, other applications would favor the fat client approach so that the capabilities of the server do not need to be upgraded.

As the use of the client/server architecture has matured, the trend is to place volatile application logic on the server. This simplifies deployment of software updates as changes are made to the application logic [PAU95].

### 28.1.4 Linking c/s Software Subsystems

A number of different mechanisms are used to link the various subsystems of the client/server architecture. These mechanisms are incorporated into the network and operating system structure and are transparent to the end-user at the client site. The most common types of linking mechanisms are:

**What options are available for linking subsystems?**

- *Pipes.* Widely used in UNIX-based systems, pipes permit messaging between different machines running on different operating systems.
- *Remote procedure calls*. These permit one process to invoke the execution of another process or module which resides on a different machine.
- *Client/server SQL interaction*. This is used to pass SQL requests and associated data from one component (typically on the client) to another component (typically the DBMS on the server). This mechanism is limited to relational database management system (RDBMS) applications.

In addition, object-oriented implementation of the c/s software subsystems results in "linkage" using an object request broker. This approach is discussed in the following section.

### 28.1.5 Middleware and Object Request Broker Architectures

**KEY POINT**

An ORB enables an object that resides on a client to send a message to a method encapsulated by an object that resides on a server.

The c/s software subsystems discussed in the preceding sections are implemented by components (objects) that must be capable of interacting with one another within a single machine (either client or server) or across the network. An *object request broker* is middleware that enables an object that resides on a client to send a message to a method that is encapsulated by an object that resides on a server. In essence, the ORB intercepts the message and handles all communication and coordination activities required to find the object to which the message was addressed, invoke its method, pass appropriate data to the object, and transfer the resulting data back to the object that generated the message in the first place.

**WebRef**

The latest information on component standards can be obtained at **www.omg.com, www.microsoft.com/COM,** and **java.sun.com/beans**

Three widely used standards that implement an object request broker philosophy—CORBA, COM, and JavaBeans—were discussed briefly in Chapter 27. CORBA will be used to illustrate the use of ORB middleware.

The basic structure of a CORBA architecture is illustrated in Figure 28.3. When CORBA is implemented in a client/server system, objects and object classes (Chapter 20) on both the client and the server are defined using an *interface description language*, a declarative language that allows a software engineer to define objects,

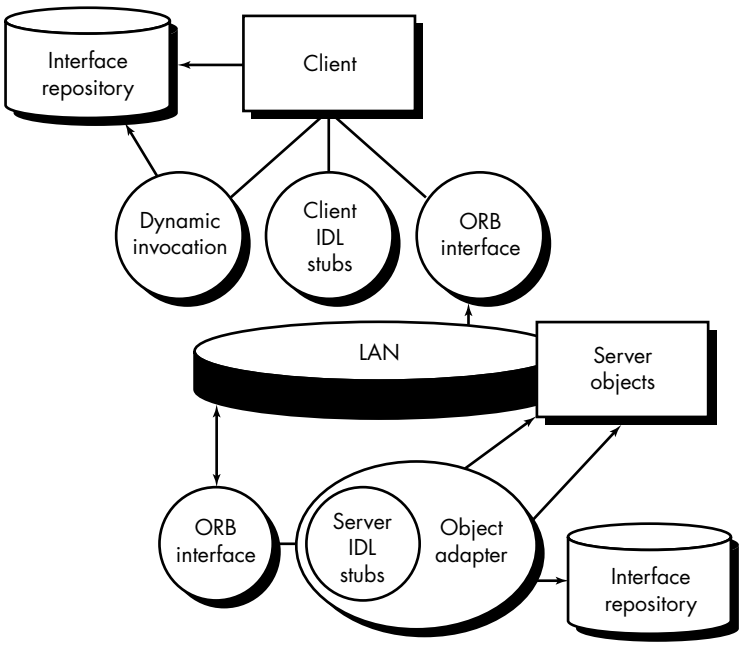



**FIGURE 28.3** The basic CORBA architecture

attributes, methods, and the messages required to invoke them. In order to accommodate a request for a server-resident method by a client-resident object, client and server IDL stubs are created. The stubs provide the gateway through which requests for objects across the c/s system are accommodated.

Because requests for objects across the network occur at run time, a mechanism for storing the object description must be established so that pertinent information about the object and its location are available when needed. The interface repository accomplishes this.

**Quote:**

"Adoption of CORBA is a positive step, but it is not enough to resolve the most critical software challenges."

**Thomas Mowbray and Raphael Malveau**

When a client application must invoke a method contained within an object elsewhere in the system, CORBA uses dynamic invocation to (1) obtain pertinent information about the desired method from the interface repository, (2) create a data structure with parameters to be passed to the object, (3) create a request for the object, and (4) invoke the request. The request is then passed to the *ORB core*—an implementation-specific part of the network operating system that manages requests—and the request is fulfilled.

The request is passed through the core and is processed by the server. At the server site, an *object adapter* stores class and object information in a server-resident interface repository, accepts and manages incoming requests from the client, and performs a variety of other object management functions [ORF99]. At the server, IDL stubs that are similar to those defined at the client machine are used as the interface to the actual object implementation resident at the server site.

Software development for a modern c/s system is object oriented. Using the CORBA architecture described briefly in this section, software developers can create an environment in which objects can be reused throughout a large network environment. For further information on CORBA and its overall impact on software engineering for c/s systems, the interested reader should refer to [HOQ99] and [SIE99].

## 28.2 SOFTWARE ENGINEERING FOR C/S SYSTEMS

A number of different software process models were introduced in Chapter 2. Although any of them could be adapted for use during the development of software for c/s systems, two approaches are most commonly used: (1) an evolutionary paradigm that makes use of event-based and/or object-oriented software engineering and (2) component-based software engineering (Chapter 27) that draws on a library of COTS and in-house software components.

Client/server systems are developed using the classic software engineering activities—analysis, design, construction, and testing—as the system evolves from a set of general business requirements to a collection of validated software components that have been implemented on client and server machines.

## 28.3 ANALYSIS MODELING ISSUES

The requirements modeling activity for c/s systems differs little from the analysis modeling methods applied to more conventional computer architectures. Therefore, the basic analysis principles discussed in Chapter 11 and the analysis modeling methods presented in Chapters 12 and 21 apply equally well to c/s software. It should be noted, however, that, because many modern c/s systems make use of reusable components, the qualification activities associated with CBSE (Chapter 27) also apply.

Because analysis modeling avoids specification of implementation detail, issues associated with the allocation of software components to client and server are considered only as the transition is made to design.[3] However, because an evolutionary approach to software engineering is applied for c/s systems, implementation decisions on the overall c/s approach (e.g., fat client vs. fat server) may be made during early analysis and design iterations.

## 28.4 DESIGN FOR C/S SYSTEMS

When software is being developed for implementation using a specific computer architecture, the design approach must consider the specific construction environment. In essence, the design should be customized to accommodate the hardware architecture.

---

3 For example, a CORBA-compliant c/s architecture (Section 28.1.5) will have a profound impact on design and implementation decisions.

When software is designed for implementation using client/server architecture, the design approach must be "customized" to accommodate the following issues:

- Data and architectural design (Chapter 14) dominate the design process. To effectively use the capabilities of a relational database management system (RDBMS) or object-oriented database management system (OODBMS) the design of the data becomes even more significant than in conventional applications.
- When the event-driven paradigm is chosen, behavioral modeling (an analysis activity, Chapters 12 and 21) should be conducted and the control-oriented aspects implied by the behavioral model should be translated into the design model.
- The user interaction/presentation component of a c/s system implements all functions that are typically associated with a graphical user interface. Therefore, interface design (Chapter 15) is elevated in importance.
- An object-oriented view of design (Chapter 22) is often chosen. Instead of the sequential structure provided by a procedural language, an object structure is provided by the linkage between an event initiated at the GUI and an event handling function within the client-based software.


*Although c/s software is different, you can use conventional or OO design methods with very few modifications.*

Although debate continues on the best analysis and design approach for c/s systems, object-oriented methods (Chapters 21 and 22) appear to have the best combination of features. However, conventional methods (Chapters 12 through 16) can also be adopted.

### 28.4.1 Architectural Design for Client/Server Systems

The architectural design of a client/server system is often characterized as a *communicating processes* style. Bass, Clements, and Kazman [BAS98] describe this architecture in the following way:

> The goal is to achieve the quality of scalability. A server exists to serve data to one or more clients, which are typically located across a network. The client originates a call to the server, which works, synchronously or asynchronously, to serve the client's request. If the server works synchronously, it returns control to the client at the same time it returns data. If the server works asynchronously, it returns only data to the client (which has its own thread of control).

**XRef**

A detailed discussion of architecture is presented in Chapter 14.

Because modern c/s systems are component based, an object request broker architecture (Figure 28.3) is used to implement this synchronous or asynchronous communication.

At the architectural level, the CORBA[4] interface description language is used to specify interface details. The use of IDL allows application software components to access ORB services (components) without knowledge of their internal workings.

4 An analogous approach is used in COM and JavaBeans.

The ORB also has the responsibility for coordinating communication among components for both the client and server. To accomplish this, the designer specifies an *object adapter* (also called a *wrapper*) that provides the following services [BAS98]:

- Component (object) implementations are registered.
- All component (object) references are interpreted and reconciled.
- Component (object) references are mapped to the corresponding component implementation.
- Objects are activated and deactivated.
- Methods (operations) are invoked when messages are transmitted.
- Security features are implemented.

To accommodate COTS components supplied by different vendors and in-house components that may have been implemented using different technologies, the ORB architecture must be designed to achieve interoperability among components. To accomplish this CORBA uses a bridging concept.

Assume that a client has been implemented using ORB protocol *X* and the server has been implemented using ORB protocol *Y*. Both protocols are CORBA compliant, but because of internal implementation differences, they must communicate to a "bridge" that provides a mechanism for translation between internal protocols [BAS98]. The bridge translates messages so that client and server can communicate smoothly.

### 28.4.2 Conventional Design Approaches for Application Software

In client/server systems, the data flow diagram (Chapters 12 and 14) can be used to establish the scope of a system, identify the high-level functions and subject data areas (data stores), and permit the decomposition of the high-level functions. In a departure from the traditional DFD approach, however, decomposition stops at the level of an elementary business process rather than continuing to the level of an atomic process.

In the c/s context, an *elementary business process* (EBP) can be defined as a set of tasks performed without a break by one user at a client site. The tasks are either performed fully or not at all.

The entity relationship diagram also assumes an expanded role. It continues to be used to decompose the subject data areas (data stores) of the DFD in order to establish a high-level view of a database that is to be implemented using an RDBMS. Its new role is to provide the structure for defining high-level business objects (Section 28.4.3).

Instead of serving as a tool for functional decomposition, the structure chart is now used as an assembly diagram to show the components involved in the solution

for an elementary business process. These components, consisting of interface objects, application objects, and database objects, establish how the data are to be processed.

### 28.4.3 Database Design

Database design is used to define and then specify the structure of business objects used in the client/server system. The analysis required to identify business objects is accomplished using business process engineering methods discussed in Chapter 10. Conventional analysis modeling notation (Chapter 12), such as the ERD, can be used to define business objects, but a database repository should be established in order to capture the additional information that cannot be fully documented using a graphic notation such as an ERD.

In this repository, a *business object* is defined as information that is visible to the purchasers and users of the system, not its implementers. This information, implemented using a relational database, can be maintained in a design repository. The following design information is collected for the client/server database [POR94]:

- *Entities* are identified within the ERD for the new system.
- *Files* implement the entities identified within the ERD.
- *File-to-field relationship*s establish the layout for the files by identifying which fields are included in which files.
- *Fields* define the fields in the design (the data dictionary).
- *File-to-file relationships* identify related files that can be joined to create logical views or queries.
- *Relationship validation* identifies the type of file-to-file or file-to-field relationships used for validation.
- *Field type* is used to permit inheritance of field characteristics from field superclasses (e.g., date, text, number, value, price).
- *Data type* specifies the characteristics of the data contained in a field.
- *File type* is used to identify the location of the file.
- *Field functions* include key, foreign key, attribute, virtual field, derived field, and the like.
- *Allowed values* identify values allowed for status type fields.
- *Business rules* are the rules for editing, calculating derived fields, and so on.

**Quote:**

"The organization of data in a database has to represent the underlying meaning or semantics of the data correctly and efficiently."

**Gio Wiederhold**

The trend toward distributed data management has accelerated as c/s architectures have become more pervasive. In c/s systems that implement this approach, the data management component resides on both the client and the server. Within the context of database design, a key issue is data distribution. That is, how are data distributed between the client and server and dispersed across the nodes of a network?

A relational database system enables easy access to distributed data through the use of structured query language. The advantage of SQL in a c/s architecture is that it is "nonnavigational" [BER92]. In an RDBMS, the type of data is specified using SQL, but no navigational information is required. Of course, the implication of this is that the RDBMS must be sophisticated enough to maintain the location of all data and be capable of defining the best path to it. In less sophisticated database systems, a request for data must indicate what is to be accessed and where it is. If application software must maintain navigational information, data management becomes much more complicated for c/s systems.

It should be noted that other data distribution and management techniques are also available to the designer [BER92]:

**What options exist for distributing data within a c/s system?**

**Manual extract.** The user is allowed to manually copy appropriate data from a server to a client. This approach is useful when static data are required by a user and the control of the extract can be left in the user's hands.

**Snapshot.** This technique automates the manual extract by specifying a "snapshot" of data that should be transferred from a server to a client at predefined intervals. This approach is useful for distributing relatively static data that require only infrequent update.

**Replication.** This technique can be used when multiple copies of data must be maintained at different sites (e.g., different servers or clients and servers). Here, the level of complexity escalates because data consistency, updates, security, and processing must all be coordinated at multiple sites.

**Fragmentation.** In this approach, the system database is fragmented across multiple machines. Although intriguing in theory, fragmentation is exceptionally difficult to implement and is not encountered frequently.

Database design and, more specifically, database design for c/s systems are topics that are beyond the scope of this book. The interested reader should see [BRO91], [BER92], [VAS93], and [ORF99] for additional discussion.

### 28.4.4 An Overview of a Design Approach

Porter [POR95] suggests a set of steps for designing an elementary business process that combines elements of conventional design with elements of object-oriented design. It is assumed that a requirements model which defines business objects has been developed and refined prior to the start of the design of elementary business processes. The following steps are then used to derive the design:

1. For each elementary business process, identify the files that are created, updated, referenced, or deleted.
2. Use the files identified in step 1 as the basis for defining components or objects.

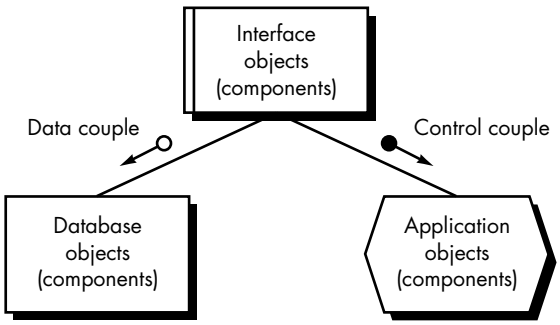



**FIGURE 28.4** Structure chart notation for c/s components

3. For each component, retrieve the business rules and other business object information that has been established for the relevant file.
4. Determine which rules are relevant to the process, and decompose the rules down to a method level.
5. As required, define any additional components that are needed to implement the methods.

Porter [POR95] suggests a structure chart notation (Figure 28.4) for representing the component structure of an elementary business process. However, a different symbology is used so that the chart will conform to the object-oriented nature of c/s software. Referring to the figure, five different symbols are encountered:

**Interface object.** This type of component, also called the *user interaction/presentation component,* is typically built over a single file and related files that have been joined through a query. It includes methods for formatting the GUI interface and client-resident related application logic. It also includes embedded SQL that specifies database processing performed on the primary file over which the interface is built. If application logic normally associated with an interface object is implemented on a server instead, typically through the use of the middleware tools, the application logic operating on the server should be identified as a separate application object.

**Database object.** This type of component is used to identify database processing such as record creation or selection based on a file other than the primary file over which an interface object is built. It should be noted that, if the primary file over which an interface object is built is processed in a different manner, using a second SQL statement to retrieve a file in an alternate sequence. For example, the second file processing technique should be identified separately on the structure chart as a separate database object.

**Application object.** Used by either an interface object or a database object, this component is invoked by either a database trigger or a remote procedure call. It can also be used to identify business logic normally associated with interface processing that has been moved to the server for operation.

**Data couple.** When one object invokes another independent object, a message is passed between the two objects. The data couple symbol is used to denote this occurrence.

**Control couple.** When one object invokes another independent object and no data are passed between the two objects, a control couple symbol is used.

### 28.4.5 Process Design Iteration

The design repository (Section 28.4.3) used to represent business objects is also used to represent interface, application, and database objects. The following entities are identified:

- *Methods* describe how a business rule is to be implemented.
- *Elementary processes* define the elementary business processes identified in the analysis model.
- *Process/component link* identifies the components that make up the solution for an elementary business process.
- *Components* describe the components shown on the structure chart.
- *Business rule/component link* identifies the components that are significant to the implementation of a given business rule.

If a repository is implemented using an RDBMS, the designer will have access to a useful design tool that provides reporting to aid both construction and future maintenance of a c/s system.

## 28.5 TESTING ISSUES[5]

The distributed nature of client/server systems pose a set of unique problems for software testers. Binder [BIN92] suggests the following areas of focus:


Useful c/s testing information and resources are presented at **www.icon-stl.net/~djmosley/**

- Client GUI considerations.
- Target environment and platform diversity considerations.
- Distributed database considerations (including replicated data).
- Distributed processing considerations (including replicated processes).
- Nonrobust target environment.
- Nonlinear performance relationships.

The strategy and tactics associated with c/s testing must be designed in a manner that allows each of these issues to be addressed.

---

5 This section is a much abbreviated and adapted version of an unpublished paper written by Daniel Mosley (used with the author's permission). An updated an expanded discussion can be found in [MOS99].

### 28.5.1 Overall c/s Testing Strategy

In general, the testing of client/server software occurs at three different levels: (1) individual client applications are tested in a "disconnected" mode, the operation of the server and the underlying network are not considered; (2) the client software and associated server applications are tested in concert, but network operations are not explicitly exercised; (3) the complete c/s architecture, including network operation and performance, is tested.

Although many different types of tests are conducted at each of these levels of detail, the following testing approaches are commonly encountered for c/s applications:

**? What types of tests are conducted for c/s systems?**

**Application function tests.** The functionality of client applications is tested using the methods discussed in Chapter 17. In essence, the application is tested in stand-alone fashion in an attempt to uncover errors in its operation.

**Server tests.** The coordination and data management functions of the server are tested. Server performance (overall response time and data throughput) is also considered.

**Database tests.** The accuracy and integrity of data stored by the server is tested. Transactions posted by client applications are examined to ensure that data are properly stored, updated, and retrieved. Archiving is also tested.

**Transaction tests.** A series of tests are created to ensure that each class of transactions is processed according to requirements. Tests focus on the correctness of processing and also on performance issues (e.g., transaction processing times and transaction volume).

**Network communication tests.** These tests verify that communication among the nodes of the network occurs correctly and that message passing, transactions, and related network traffic occur without error. Network security tests may also be conducted as part of these tests.

**XRef**

Requirements elicitation techniques and use-cases are discussed in Chapter 11.

To accomplish these testing approaches, Musa [MUS93] recommends the development of operational profiles derived from client/server usage scenarios. An *operational profile* indicates how different types of users interoperate with the c/s system. That is, the profiles provide a "pattern of usage" that can be applied when tests are designed and executed. For example, for a particular type of user, what percentage of transactions will be inquiries? updates? orders?

To develop the operational profile, it is necessary to derive a set of user scenarios [BIN95] that are similar to use-cases discussed earlier in this book. Each scenario addresses who, where, what, and why. That is, who the user is, where (in the physical c/s architecture) the system interaction occurs, what the transaction is, and why it has occurred. Scenarios can be derived using requirements elicitation techniques or through less formal discussions with end-users. The result, however, should be

the same. Each scenario should provide an indication of the system functions that will be required to service a particular user, the order in which those functions are required, the timing and response that is expected, and the frequency with which each function is used. These data are then combined (for all users) to create the operational profile.

**Quote:**

"The topic of testing is one area in which a good deal of commonality exists between traditional systems and client/server-based systems."

**Kelley Bourne**

The strategy for testing c/s architectures is analogous to the testing strategy for software-based systems described in Chapter 18. Testing begins with testing in the small. That is, a single client application is tested. Integration of the clients, the server, and the network are tested progressively. Finally, the entire system is tested as an operational entity.

Traditional testing views module/subsystem/system integration and testing (Chapter 18) as top down, bottom up, or some variation of the two. Module integration in c/s development may have some top-down or bottom-up aspects, but integration in c/s projects tends more toward parallel development and integration of modules across all design levels. Therefore, integration testing in c/s projects is sometimes best accomplished using a nonincremental or "big bang" approach.

The fact that the system is not being built to use prespecified hardware and software affects system testing. The networked cross-platform nature of c/s systems requires that we pay considerably more attention to configuration testing and compatibility testing.

Configuration testing doctrine forces testing of the system in all of the known hardware and software environments in which it will operate. Compatibility testing ensures a functionally consistent interface across hardware and software platforms. For example, a Windows-type interface may be visually different depending on the implementation environment, but the same basic user behaviors should produce the same results regardless of the client interface standard.

### 28.5.2 c/s Testing Tactics

Even if the c/s system has not been implemented using object technology, object-oriented testing techniques (Chapter 23) make good sense because the replicated data and processes can be organized into classes of objects that share the same set of properties. Once test cases have been derived for a class of objects (or their equivalent in a conventionally developed system), those test cases should be broadly applicable for all instances of the class.

The OO point of view is particularly valuable when the graphical user interface of modern c/s systems is considered. The GUI is inherently object oriented and departs from traditional interfaces because it must operate on many platforms. In addition, testing must explore a large number of logic paths because the GUI creates, manipulates, and modifies a broad range of graphical objects. Testing is further complicated because the objects can be present or absent, they may exist for a length of time, and they can appear anywhere on the desktop.

What this means is that the traditional capture/playback approach for testing conventional character-based interfaces must be modified in order to handle the complexities of the GUI environment. A functional variation of the capture/playback paradigm called *structured capture/playback* [FAR93] has evolved for GUI testing.

Traditional capture/playback records input as keystrokes and output as screen images, which are saved and compared against inputs and output images of subsequent tests. Structured capture/playback is based on an internal (logical) view of external activities. The application program's interactions with the GUI are recorded as internal events, which can be saved as "scripts" written in Microsoft's Visual Basic, one of the C variants, or in the vendor's proprietary language.

Tools that exercise GUIs do not address traditional data validation or path testing needs. The black-box and white-box testing methods discussed in Chapter 17 are applicable in many instances and the special object-oriented tactics presented in Chapter 23 are appropriate for both client and server software.

## 28.6 SUMMARY

Although client/server systems can adopt one or more of the software process models and many of the analysis, design, and testing methods described earlier in this book, the special architectural features of c/s require customization of the software engineering approach. In general, the software process model applied for c/s systems is evolutionary in nature and the technical methods often gravitate toward object-oriented or component-based approaches. The developer must describe objects that result in the implementation of user interaction/presentation, database, and application subsystems. The components (objects) defined for these subsystems must be allocated to either the client or server machines and can be linked via an object request broker.

Object request broker architectures support c/s designs in which client objects send messages to server objects. The CORBA standard makes use of interface definition language, and interface repositories manage requests for objects regardless of their location on the network.

Analysis and design for client/server systems make use of data flow and entity relationship diagrams, modified structure charts, and other notation that is encountered in the development of conventional applications. Testing strategies must be modified to accommodate tests that examine network communication and the interplay between software that resides on client and server.

## REFERENCES

[BAS98] Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice,* Addison-Wesley, 1998.

[BER92] Berson, Alex, *Client/Server Architecture,* McGraw-Hill, 1992.

[BIN92] Binder, R., "A CASE-Based Systems Engineering Approach to Client-Server Development," *CASE Trends,* 1992.
[BIN95] Binder, R., "Scenario-Based Testing for Client Server Systems," *Software Development,* vol. 3, no. 8, August 1995, pp. 43–49.
[BRO91] Brown, A.W., *Object-Oriented Databases,* McGraw-Hill, 1991.
[FAR93] Farley, K.J., "Software Testing for Windows Developers," *Data Based Advisor,* November 1993, pp. 45–46, 50–52.
[HOQ99] Hoque, R., *CORBA for Real Programmers,* Academic Press/Morgan Kaufmann, 1999.
[ORF99] Orfali, R., D. Harkey, and J. Edwards, *Essential Client/Server Survival Guide,* 3rd ed., Wiley, 1999.
[MOS99] Mosley, D., *Client Server Software Testing on the Desk Top and the Web,* Prentice-Hall, 1999.
[MUS93] Musa, J., "Operational Profiles in Software Reliability Engineering," *IEEE Software,* March 1993, pp. 14–32.
[PAU95] L.G. Paul, "Client/Server Deployment," *Computerworld,* December 18, 1995.
[POR94] Porter, J., *O-DES Design Manual,* Fairfield University, 1994.
[POR95] Porter, J., *Synon Developer's Guide,* McGraw-Hill, 1995.
[SIE99] Siegel, J., *CORBA 3 Fundamentals and Programming,* Wiley, 1999.
[VAS93] Vaskevitch, D., *Client/Server Strategies,* IDG Books, 1993.

## PROBLEMS AND POINTS TO PONDER

**28.1.** Using trade publications or Internet resources for background information, define a set of criteria for evaluating tools for c/s software engineering.

**28.2.** Suggest five applications in which a fat server would seem to be an appropriate design strategy.

**28.3.** Suggest five applications in which a fat client would seem to be an appropriate design strategy.

**28.4.** Do some additional research on the CORBA standard and determine how the latest release of the standard addresses interoperability among different ORBs provided by different vendors.

**28.5.** Research a structured query language and provide a brief example of how a transaction might be characterized using the language.

**28.6.** Research the latest advances in groupware and develop a brief presentation for your class. Your instructor may assign a specific function to different presenters.

**28.7.** A company is establishing a new e-commerce division to sell casual apparel and outdoor merchandise. The e-catalog will be published on the World Wide Web and orders can be placed via the Web site, by e-mail, or via telephone or fax. A

client/server system will be built to support order processing at the company site. Define a set of high-level objects that would be required for the order-processing system and organize these objects into three component categories: the user interaction/presentation, database, and application.

**28.8.** Define business rules to establish when a shipment can be made if payment is by credit card for the system described in Problem 28.7. Add additional rules if payment is by check.

**28.9.** Develop a state transition diagram (Chapter 12) that defines the events and states that would be visible to an order entry clerk working at a client PC within the e-commerce sales division (Problem 28.7).

**28.10.** Provide examples of three or four messages that might result in a request from a client for a method maintained on the server.

## FURTHER READINGS AND INFORMATION SOURCES

Although software engineering methods for client/server systems are quite similar to conventional and OO systems, specialized knowledge and techniques are required. Worthwhile introductions to basic concepts have been written by Lowe and Helda (*Client/Server Computing for Dummies,* 3rd ed., IDG Books Worldwide, 1999) and Zantinge and Adriaans (*Managing Client/Server,* Addison-Wesley, 1997). At an intermediate level, McClanahan (*Developing Client-Server Applications,* IDG Books Worldwide, 1999) covers a broad range of c/s topics. On a more sophisticated level, Orfali and his colleagues [ORF99] and Linthicum (*Guide to Client/Server and Intranet Development,* Wiley, 1997) provide detailed guidelines for engineering c/s applications. Berson (*Client/Server Architecture,* 2nd ed., McGraw-Hill, 1996) discusses component and architecture issues.

Network computers have become a hot technology topic (and a risky business strategy) in recent years. Sinclair and Merkow (*Thin Clients Clearly Explained,* Morgan Kaufmann, 1999), Friedrichs and Jubin (*Java Thin-Client Programming for a Network Computing Environment,* Prentice-Hall, 1999), Dewire (*Thin Clients,* McGraw-Hill, 1998), and Kanter (*Understanding Thin-Client/Server Computing,* Microsoft Press, 1998) provide worthwhile guidance on how to design, build, deploy, and support thin-client systems.

Beginning with the modeling of business events, Ruble (*Practical Analysis and Design for Client/Server and GUI Systems,* Yourdon Press, 1997) provides an in-depth discussion of techniques for the analysis and design of c/s systems. Books by Goldman, Rawles, and Mariga (*Client/Server Information Systems: A Business-Oriented Approach,* Wiley, 1999); Shan, Earle, and Lenzi (*Enterprise Computing with Objects: From Client/Server Environments to the Internet,* Addison-Wesley, 1997); and Gold-Bernstein

and Marca (*Designing Enterprise Client/Server Systems,* Prentice-Hall, 1997) consider c/s in a broader enterprise context.

Loosley and Douglas (*High-Performance Client/Server,* Wiley, 1997) explain the principles of software performance engineering and apply them to distributed systems architecture and design. Heinckiens and Loomis (*Building Scalable Database Applications: Object-Oriented Design, Architectures, and Implementations,* Addison-Wesley, 1998) emphasize database design in their guide for building client/server applications. Ligon (*Client/Server Communications Services: A Guide for the Applications Developer,* McGraw-Hill, 1997) considers a wide variety of communication-related topics including TCP/IP, ATM, EDI, CORBA, messaging, and encryption. Schneberger (*Client/Server Software Maintenance,* McGraw-Hill, 1997) presents a framework for controlling c/s software maintenance costs and optimizing user support.

Hundreds of books address vendor-specific c/s systems development. The following represents a small sampling:

Anderson, G.W., *Client/Server Database Design with Sybase: A High-Performance and Fine-Tuning Guide,* McGraw-Hill, 1997.

Barlotta, M.J., *Distributed Application Development with Powerbuilder 6,* Manning Publications, 1998.

Bates, R.J., *Hands-on Client/Server Internetworking,* McGraw-Hill, 1997.

Mahmoud, Q.H., *Distributed Programming with Java,* Manning, 1998.

Orfali, R. and D. Harkey, *Client/Server Programming with JavaBeans,* Wiley, 1999.

Sankar, K., *Building Internet Client/Server Systems,* McGraw-Hill, 1999.

Detailed guidebooks for c/s testing have been written by Mosley [MOS99] and Bourne (*Testing Client/Server Systems,* McGraw-Hill, 1997). Both authors provide in-depth discussion of testing strategies, tactics, and tools.

A wide variety of information sources on client/server software engineering is available on the Internet. An up-to-date list of World Wide Web references that are relevant to c/s systems can be found at the SEPA Web site:

**http://www.mhhe.com/engcs/compsci/pressman/resources/client-server.mhtml**

CHAPTER

# 29 WEB ENGINEERING

KEY CONCEPTS



The World Wide Web and the Internet have drawn the general populace into the world of computing. We purchase stock and mutual funds, download music, view movies, get medical advice, book hotel rooms, sell personal items, schedule airline flights, meet people, do our banking, take college courses, buy groceries—we do just about anything and everything in the virtual world of the Web. Arguably, the Web and the Internet that empowers it are the most important developments in the history of computing. These computing technologies have drawn us all (with billions more who will eventually follow) into the information age. They have become integral to daily life in the first years of the twenty-first century.

For those of us who can remember a world without the Web, the chaotic growth of the technology harkens back to another era—the early days of software. It was a time of little discipline, but enormous enthusiasm and creativity. It was a time when programmers often hacked together systems—some good, some bad. The prevailing attitude seemed to be "Get it done fast, and get it into the field, we'll clean it up (and better understand what we really need to build) as we go." Sound familiar?

In a virtual round table published in *IEEE Software* [PRE98], I staked out my position with regard to Web engineering:

## QUICK LOOK

**What is it?** Web-based systems and applications (WebApps) deliver a complex array of content and functionality to a broad population of end-users. Web engineering is the process used to create high-quality WebApps. Web engineering is not a perfect clone of software engineering, but it borrows many of software engineering's fundamental concepts and principles, emphasizing the same technical and management activities. There are subtle differences in the way these activities are conducted, but an overriding philosophy that dictates a disciplined approach to the development of a computer-based system is identical.

**Who does it?** Web engineers and nontechnical content developers create the WebApp.

**Why is it important?** As WebApps become increasing integrated in business strategies for small and large companies (e.g., e-commerce), the need to build reliable, usable, and adaptable systems grows in importance. That's why a disciplined approach to WebApp development is necessary.

**What are the steps?** Like any engineering discipline, Web engineering applies a generic approach that is tempered with specialized strategies, tactics, and methods. The Web engineering process begins with a formulation of the problem to be solved by the WebApp. The project is planned, and the

**QUICK LOOK**

requirements of the WebApp are analyzed. Architectural, navigational, and interface design are conducted. The system is implemented using specialized languages and tools associated with the Web, and testing commences. Because WebApps evolve continuously, mechanisms for configuration control, quality assurance, and ongoing support are needed.

**What is the work product?** A variety of Web engineering work products (e.g., analysis models, design models, test procedures) are produced. The final output is the operational WebApp.

**How do I ensure that I've done it right?** Use the same SQA practices that are applied in every software engineering process—formal technical reviews assess the analysis and design models, specialized reviews consider usability; testing is applied to uncover errors in content, functionality, and compatibility.

It seems to me that just about any important product or system is worth engineering. Before you start building it, you'd better understand the problem, design a workable solution, implement it in a solid way, and test it thoroughly. You should probably also control changes to it as you work and have some mechanism for ensuring the end result's quality. Many Web developers don't argue with this; they just think their world is really different and that conventional software engineering approaches simply don't apply.

This leads us to a pivotal question: Can software engineering principles, concepts, and methods be applied to Web development? Many of them can, but their application may require a somewhat different spin.

**Quote:**

"The engineering principles of planning before designing and designing before building have withstood every prior technology transition; they'll survive this transition as well."

**Watts Humphrey**

But what if the current ad hoc approach to Web development persists? In the absence of a disciplined process for developing Web-based systems, there is increasing concern that we may face serious problems in the successful development, deployment, and "maintenance" of these systems. In essence, the application infrastructure that we are creating today may lead to something that might be called a *tangled Web* as we move further into this new century. This phrase connotes a morass of poorly developed Web-based applications that have too high a probability of failure. Worse, as Web-based systems grow more complex, a failure in one can and will propagate broad-based problems across many. When this happens, confidence in the entire Internet may be shaken irreparably. Worse, it may lead to unnecessary and ill-conceived government regulation, leading to irreparable harm to these unique technologies.

In order to avoid a tangled Web and achieve greater success in development and application of large-scale, complex Web-based systems, there is a pressing need for disciplined Web engineering approaches and new methods and tools for development, deployment, and evaluation of Web-based systems and applications. Such approaches and techniques must take into account the special features of the new medium, the operational environments and scenarios, and the multiplicity of user profiles that pose additional challenges to Web-based application development.

*Web Engineering* (WebE) is concerned with the establishment and use of sound scientific, engineering, and management principles and disciplined and systematic

approaches to the successful development, deployment, and maintenance of high-quality Web-based systems and applications [MUR99].

## 29.1 THE ATTRIBUTES OF WEB-BASED APPLICATIONS

There is little debate that Web-based systems and applications[1] (we will refer to these collectively as *WebApps*) are different than the many other categories of computer software discussed in Chapter 1. Powell summarizes the primary differences when he states that Web-based systems "involve a mixture between print publishing and software development, between marketing and computing, between internal communications and external relations, and between art and technology" [POW98]. The following attributes are encountered in the vast majority of WebApps:[2]

**Network intensive.** By its nature, a WebApp is network intensive. It resides on a network and must serve the needs of a diverse community of clients. A WebApp may reside on the Internet (thereby enabling open worldwide communication). Alternatively, an application may be placed on an intranet (implementing communication across an organization) or an Extranet (internetwork communication).


WebApps are network intensive, content driven, and continuously evolving. These attributes have a profound impact on the way in which WebE is conducted.

**Content driven.** In many cases, the primary function of a WebApp is to use hypermedia to present text, graphics, audio, and video content to the end-user.

**Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically spaced releases, Web applications evolve continuously. It is not unusual for some WebApps (specifically, their content) to be updated on an hourly schedule.

Some argue that the continuous evolution of WebApps makes the work performed on them analogous to gardening. Lowe {LOW99] discusses this when he writes:

> Engineering is about adopting a consistent and scientific approach, tempered by a specific practical context, to development and commissioning of systems or applications. Web site development is often much more about creating an infrastructure (laying out the garden) and then "tending" the information which grows and blooms within this garden. Over time the garden (i.e., Web site) will continue to evolve, change, and grow. A good initial architecture should allow this growth to occur in a controlled and consistent manner . . . we could have "tree surgeons" that prune "trees" (i.e., sections of the site) within the "garden"

---

1 Included within this category are complete Web sites, specialized functionality within Web sites, and information processing applications that reside on the Internet or on an intranet or Extranet.

2 In the context of this chapter, the term *Web application* encompasses everything from a simple Web page that might help a consumer compute an automobile lease payment to a comprehensive Web site that provides complete travel services for business people and vacationers.

(the site itself) while ensuring cross-referential integrity. We could have "garden nurseries" where young plants (i.e., design patterns for Web sites) are "grown." I probably should stop this analogy before I get too carried away!

Continual care and feeding allows a Web site to grow (in robustness and importance). But, unlike a garden, Web applications must serve (and adapt to) the needs of more than the gardener. The following WebApp characteristics drive the process:


*There is little doubt that immediacy often holds sway in WebApp development, but be careful! Just because you have to get it done quickly does not mean that you have the luxury of doing a poorly engineered job. Quick and wrong are rarely an acceptable result.*

**Immediacy.** Web-based applications have an immediacy [NOR99] that is not found in any other type of software. That is, the time to market for a complete Web site can be a matter of a few days or weeks.[3] Developers must use methods for planning, analysis, design, implementation, and testing that have been adapted to the compressed time schedules required for WebApp development.

**Security.** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application. In order to protect sensitive content and provide secure modes of data transmission, strong security measures must be implemented throughout the infrastructure that supports a WebApp and within the application itself.

**Aesthetics.** An undeniable part of the appeal of a WebApp is its look and feel. When an application has been designed to market or sell products or ideas, aesthetics may have as much to do with success as technical design.

These general characteristics apply to all WebApps but with different degrees of influence. The following application categories are most commonly encountered in WebE work [DAR99]:

**How can we categorize WebApps?**

- *Informational*. Read-only content is provided with simple navigation and links.
- *Download*. A user downloads information from the appropriate server.
- *Customizable*. The user customizes content to specific needs.
- *Interaction*. Communication among a community of users occurs via chatroom, bulletin boards, or instant messaging.
- *User input*. Forms-based input is the primary mechanism for communicating need.
- *Transaction oriented*. The user makes a request (e.g., places an order) that is fulfilled by the WebApp.
- *Service oriented*. The application provides a service to the user (e.g., assists the user in determining a mortgage payment).
- *Portal*. The application channels the user to other Web content or services outside the domain of the portal application.

3 Reasonably sophisticated Web pages can be produced in only a few hours.

- *Database access*. The user queries a large database and extracts information.
- *Data warehousing*. The user queries a collection of large databases and extracts information.

The characteristics noted earlier in this section and the application categories just noted represent facts of life for Web engineers. The key is living within the constraints imposed by the characteristics and still producing a successful WebApp.

### 29.1.1 Quality Attributes

Every person who has surfed the Web or used a corporate intranet has an opinion about what makes a "good" WebApp. Individual viewpoints vary widely. Some users enjoy flashy graphics, others want simple text. Some demand copious information, others desire an abbreviated presentation. In fact, the user's perception of "goodness" (and the resultant acceptance or rejection of the WebApp as a consequence) might be more important that any technical discussion of WebApp quality.

But how is WebApp quality perceived? What attributes must be exhibited to achieve goodness in the eyes of end-users and at the same time exhibit the technical characteristics of quality that will enable a Web engineer to correct, adapt, enhance, and support the application over the long term?

In reality, all of the general characteristics of software quality discussed in Chapters 8, 19, and 24 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.

Detailed quality requirement tree for WebApps

Olsina and his colleagues [OLS99] has prepared a "quality requirement tree" that identifies a set of attributes that lead to high-quality WebApps. Figure 29.1 summarizes their work.

### 29.1.2 The Technologies

The design and implementation of Web-based systems and applications incorporates three important enabling technologies: component-based development, security, and Internet standards. A Web engineer must be familiar with all three in order to build high-quality WebApps.

#### Component-Based Development

The component technologies discussed in Chapters 27 and 28 have evolved in large part because of the explosive growth of Web-based systems and applications. Recalling our discussion from the preceding chapters, three major infrastructure standards are available for Web engineers: CORBA, COM/DCOM, and JavaBeans. These standards (accompanied by prebuilt components, tools, and techniques) provide an infrastructure that enables developers to deploy third party and custom developed components and allow them to communicate with one another and with system-level services.

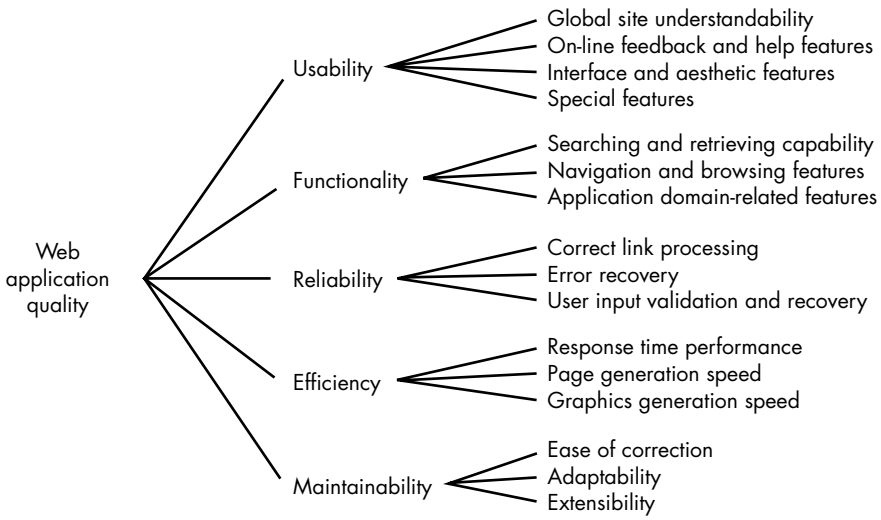



**FIGURE 29.1** Quality requirements tree [OLS99]

### Security

If a WebApp resides on a network, it is open to unauthorized access. In some cases, unauthorized access may be attempted by internal personnel. In others, outsiders (hackers) may attempt access for sport, for profit, or with more malevolent intent. A variety of security measures are provided by the network infrastructure, encryption techniques, firewalls, and other measures. A comprehensive discussion of this important topic is beyond the scope of this book. For more information, the interested reader should see [ATK97], [KAE99], and [BRE99].

**Quote:**

"The Internet is a risky place to conduct business or store assets. Hackers, crackers, snoops, spoofers, spammers, scammers, shammers, jammers, intruders, thieves, purloiners, conspirators, vandals, Trojan horse dealers, virus launchers and rouge program purveyors run loose."

**Dorothy Denning and Peter Denning**

### Internet Standards

For the last decade the dominant standard for the creation of WebApp content and structure has been HTML, a markup language that enables the developer to provide a series of tags that describe the appearance of a wide array of data objects (text, graphics, audio/video, forms, etc.). However, as the size and complexity of applications grow, a new standard—XML— has been adopted for the next generation of WebApps. XML (extensible markup language) is a strictly defined subset of the meta-language SGML [BRA97], allowing developers to define custom tags within Web page descriptions. Using an XML meta-language description, the meaning of the custom tags is defined in the information transmitted to the client site. For more information on XML, the interested reader should see [PAR99] and [STL99].

## 29.2 THE WEBE PROCESS

The characteristics of Web-based systems and applications have a profound influence on the WebE process. Immediacy and continuous evolution dictate an iter-


WebE demands an evolutionary, incremental software process.

ative, incremental process model (Chapter 2) that produces WebApp releases in rapid fire sequence. The network-intensive nature of applications in this domain suggests a population of users that is diverse (thereby making special demands on requirements elicitation and modeling) and an application architecture that can be highly specialized (thereby making demands on design). Because WebApps are often content driven with an emphasis on aesthetics, it is likely that parallel development activities will be scheduled within the WebE process and involve a team of both technical and non-technical people (e.g., copywriters, graphic designers).

## 29.3 A FRAMEWORK FOR WEBE

As WebApps evolve from static, content-directed information sources to dynamic, user-directed application environments, the need to apply solid management and engineering principles grows in importance. To accomplish this, it is necessary to develop a WebE framework that encompasses an effective process model, populated by framework activities[4] and engineering tasks. A process model for WebE is suggested in Figure 29.2.

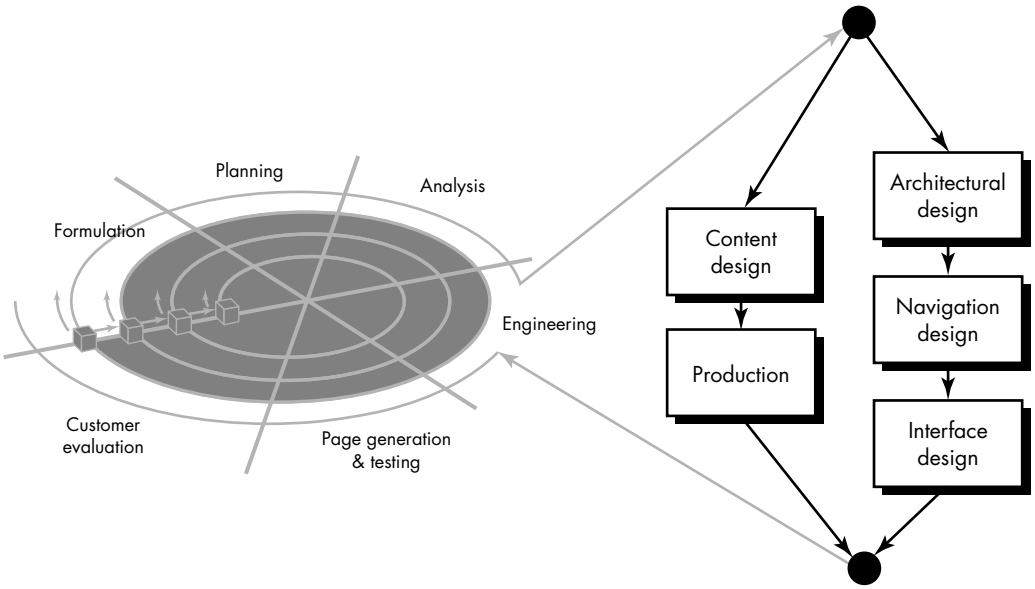



**FIGURE 29.2** The WebE process model

4 Recalling the discussion of process models in Chapter 2, framework activities are performed for all WebApps, while engineering tasks are adapted to the size and complexity of the WebApp to be developed.

The WebE process begins with a *formulation*—an activity that identifies the goals and objectives of the WebApp and establishes the scope for the first increment. *Planning* estimates overall project cost, evaluates risks associated with the development effort, and defines a finely granulated development schedule for the initial WebApp increment, with a more coarsely granulated schedule for subsequent increments. *Analysis* establishes technical requirements for the WebApp and identifies the content items that will be incorporated. Requirements for graphic design (aesthetics) are also defined.

The *engineering* activity incorporates two parallel tasks illustrated on the right side of Figure 29.2. *Content design* and *production* are tasks performed by nontechnical members of the WebE team. The intent of these tasks is to design, produce, and/or acquire all text, graphics, audio, and video content that are to become integrated into the WebApp. At the same time, a set of technical design tasks (Section 29.5) are conducted.


W3C, an industry consortium that provides access to WWW information of interest to Web engineers can be accessed at **www.w3.org**

*Page generation* is a construction activity that makes heavy use of automated tools for WebApp creation. The content defined in the engineering activity is merged with the architectural, navigation, and interface designs to produce executable Web pages in HTML, XML, and other process-oriented languages (e.g., Java). Integration with component middleware (i.e., CORBA, DCOM, or JavaBeans) is also accomplished during this activity. *Testing* exercises WebApp navigation; attempts to uncover errors in applets, scripts, and forms; and helps ensure that the WebApp will operate correctly in different environments (e.g., with different browsers).

Each increment produced as part of the WebE process is reviewed during *customer evaluation*. This is the point at which changes are requested (scope extensions occur). These changes are integrated into the next path through the incremental process flow.

## 29.4 FORMULATING/ANALYZING WEB-BASED SYSTEMS

Formulation and analysis of Web-based systems and applications represent a sequence of Web engineering activities that begins with the identification of the overall goals for a WebApp and terminates with the development of an analysis model or requirements specification for the system. Formulation allows the customer and the developer to establish a common set of goals and objectives for the construction of the WebApp. It also identifies the scope of the development effort and provides a means for determining a successful outcome. Analysis is a technical activity that identifies the data, functional, and behavioral requirements for the WebApp.

### 29.4.1 Formulation

Powell [POW98] suggests a set of questions that should be answered at the beginning of the formulation step:

**What questions should be asked to formulate the problem?**

- What is the main motivation for the WebApp?
- Why is the WebApp needed?
- Who will use the WebApp?

The answer to each of these simple questions should be stated as succinctly as possible. For example, assume that the manufacturer of home security systems has decided to establish an e-commerce Web site to sell its products directly to consumers. A statement describing the motivation for the WebApp might be

> SafeHomeInc.com[5] will allow consumers to configure and purchase all components required to install a home/business security system.

It is important to note that detail is not provided in this statement. The objective is to bound the overall intent of the site.

After discussion with various constituencies within SafeHome Inc., an answer to the second question is stated:

> SafeHomeInc.com will allow us to sell directly to consumers, thereby eliminating retailer costs and improving our profit margins. It will also allow us to increase sales by a projected 25 percent over current annual sales and will allow us to penetrate geographic regions where we currently do not have sales outlets.

Finally, the company defines the demographic for the WebApp: "Projected users of SafeHomeInc.com are homeowners and owners of small businesses."

**KEY POINT**

Both informational and applicative goals should be defined for every WebApp.

These answers imply specific goals for the SafeHomeInc.com Web site. In general, two categories of goals [GNA99] are identified:

- *Informational goals*. Indicate an intention to provide specific content and/or information to the end-user.
- *Applicative goals*. Indicate the ability to perform some task within the WebApp.

In the content of the SafeHomeInc.com WebApp, one informational goal might be

> The site will provide users with detailed product specifications, including technical description, installation instructions, and pricing information.

Examination of the answers to the questions just posed might lead to the statement of an applicative goal:

> SafeHomeInc.com will query the user about the facility (i.e., house, office/retail space) that is to be protected and make customized recommendations about the product and configuration to be used.

Once all informational and applicative goals have been identified, a user profile is developed. The user profile captures "relevant features related to potential users

5 *SafeHome* has been used as a continuing example earlier in this book.

including their background, knowledge, preferences and even more" [GNA99]. In the case of SafeHomeInc.com, a user profile would identify the characteristics of a typical purchaser of security systems (this information would be supplied by the SafeHome Inc. marketing department).

Once goals and user profiles have been developed, the formulation activity focuses on a statement of scope (Chapter 5) for the WebApp. In many cases, the goals already developed are integrated into the statement of scope. In addition, however, it is useful to indicate the degree of integration to be expected of the WebApp. That is, it is often necessary to integrate existing information systems (e.g., an existing database application) with a Web-based front end. Connectivity issues are considered at this stage.

### 29.4.2 Analysis

The concepts and principles discussed for software requirements analysis (Chapter 11) apply without revision for the Web engineering analysis activity. Scope defined during the formulation activity is elaborated to create a complete analysis model for the WebApp. Four different types of analysis are conducted during WebE:

**Content analysis.** The full spectrum of content to be provided by the WebApp is identified. Content includes text, graphics and images, video and audio data. Data modeling (Chapter 12) can be used to identify and describe each of the data objects to be used within the WebApp.

**Interaction analysis.** The manner in which the user interacts with the WebApp is described in detail. Use-cases (Chapter 11) can be developed to provide detailed descriptions of this interaction.

**Functional analysis.** The usage scenarios (use-cases) created as part of interaction analysis define the operations that will be applied to WebApp content and imply other processing functions. All operations and functions are described in detail.

**Configuration analysis.** The environment and infrastructure in which the WebApp resides are described in detail. The WebApp can reside on the Internet, an intranet, or an Extranet. In addition, the infrastructure (i.e., the component infrastructure and the degree to which a database will be used to generate content) for the WebApp should be identified at this stage.

**Quote:**

"Successful knowledge products [WebApps] allow customers to meet their needs better, faster, or cheaper themselves, rather than working through employee end-users. The Internet's ability to connect customers directly with companies provides an infrastructure for knowledge products."

**Mark McDonald**

Although a detailed requirements specification is recommended for large, complex WebApps, such documents are rare. It can be argued that the continuous evolution of WebApp requirements may make obsolete any document before it is finished. Although this may be true in the extreme, it is necessary to define an analysis model that can serve as a foundation for the design activity that follows. At a minimum, the information collected during the preceding four analysis tasks should be reviewed, modified as required, and then organized into a document that can be passed to WebApp designers.

## 29.5 DESIGN FOR WEB-BASED APPLICATIONS

**WebRef**

A worthwhile source of practical guidelines for Web site design can be found at **www.ibm.com/ibm/easy/design/lower/f060100.html**

The immediate nature of Web-based applications coupled with the pressure for continuous evolution forces a Web engineer to establish a design that solves the immediate business problem while at the same time defining an application architecture that has the ability to evolve rapidly over time. The problem, of course, is that solving the immediate problem (quickly) can result in compromises that affect the ability of the application to evolve over time. This is the designer's dilemma.

In order to perform Web-based design effectively, a Web engineer should work to reuse four technical elements [NAN98]:

**Design principles and methods.** It is important to note that the design concepts and principles discussed in Chapter 13 apply to all WebApps. Effective modularity (exhibited by high cohesion and low coupling), information hiding, stepwise elaboration, and other software design heuristics lead to Web-based systems and applications that are easier to adapt, enhance, test, and use.

Design methods for object-oriented systems discussed earlier in this book can be reused when Web-applications are created. Hypermedia defines "objects" that interact via a communication protocol that is loosely analogous to messaging. In fact the diagrammatic notation proposed for UML (Chapters 21 and 22) can be adapted for use in design activities for WebApps. In addition, a variety of hypermedia design methods have been proposed (e.g., [ISA95], [SCH96]).

**Golden rules.** Interactive hypermedia applications (WebApps) have been constructed for more than a decade. Over that time, designers have developed a set of design heuristics (*golden rules*) that can be reapplied during the design of new applications.

**Design patterns.** As we noted earlier in this book, design patterns are a generic approach for solving some small problems that can be adapted to a much wider variety of specific problems. In the context of WebApps, design patterns can be applied not only to the functional elements of an application, but to documents, graphics, and general aesthetics for a Web site.

**Templates.** A template can be used to provide a skeletal framework for any design pattern or document that is to be used within a WebApp. Nanard and Kahn [NAN98] describe this reusable design element in the following way:

> Once a template is specified, any part of a hypermedia structure that conforms to this template can be automatically generated or updated just by calling the template with relevant data [to flesh out the skeleton]. The use of constructive templates implicitly relies on the separation of hypermedia document contents from the specification of its presentation: source data are mapped into the hypertext structure as specified in the template.

**Quote:**

"To some, Web design focuses on visual look and feel . . . To others, Web design is about the structuring of information and the navigation through the document space. Others might even consider Web design to be about the technology used to build interactive Web applications. In reality, design includes all of these things and maybe more."

**Thomas Powell**

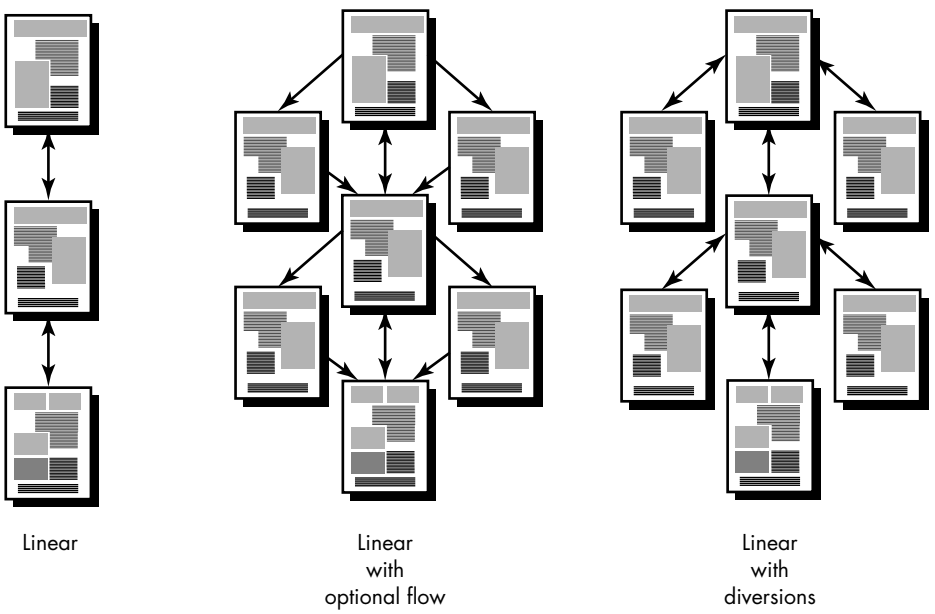



**FIGURE 29.3** Linear structures

Each of the four reusable design elements is discussed in greater detail in the sections that follow.


*Most WebApp structures just happen. Avoid this trap. Layout the WebApp structural design explicitly before you begin developing navigation or page detail.*

### 29.5.1 Architectural Design

Architectural design for Web-based systems and applications focuses on the definition of the overall hypermedia structure of the WebApp and the application of design patterns and constructive templates to populate the structure (and achieve reuse). A parallel activity, called *content design*,[6] derives the overall structure and detailed layout of the information content that will be presented as part of the WebApp.

#### WebApp Structures

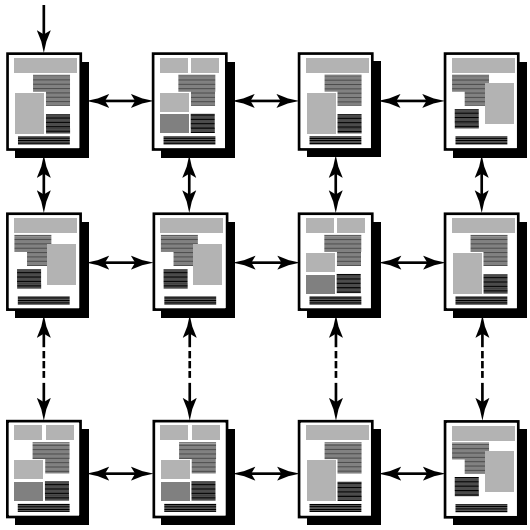
**? What structural options are available for WebApp design?**

Overall architectural structure is tied to the goals established for a WebApp, the content to be presented, the users who will visit, and the navigation philosophy (Section 29.5.3) that has been established. The architectural designer can choose from four different structures [POW98] when developing the design for a typical WebApp.

*Linear structures* (Figure 29.3) are encountered when a predictable sequence of interactions (with some variation or diversion) is common. A classic example might be a tutorial presentation in which pages of information along with related graphics, short videos, or audio are presented only after prerequisite information has been pre-

6 Content design is a nontechnical activity that is performed by copywriters, artists, graphic designers, and others who generate Web-based content. For further detail, see [DIN98] and [LYN99].

**FIGURE 29.4**
Grid Structure



sented. The sequence of content presentation is predefined and generally linear. Another example might be a product order entry sequence in which specific information must be specified in a specific order. In such cases, the structures shown in Figure 29.3 are appropriate. As content and processing become more complex, the purely linear flow shown on the left of the figure gives way to more sophisticated linear structures in which alternative content may be invoked or a diversion to acquire complementary content (structure shown on the right side of Figure 29.3) occurs.


Grid structures work well when content can be organized categorically in two or more dimensions.

*Grid structures* (Figure 29.4) are an architectural option that can be applied when WebApp content can be organized categorically in two (or more) dimensions. For example, consider a situation in which an e-commerce site sells golf clubs. The horizontal dimension of the grid represents the type of club to be sold (e.g., woods, irons, wedges, putters). The vertical dimension represents the offerings provided by various golf club manufacturers. Hence, a user might navigate the grid horizontally to find the putters column and then vertically to examine the offerings provided by those manufacturers that sell putters. This WebApp architecture is useful only when highly regular content is encountered [POW98].


*Coupling is a tricky issue for WebApp architectures. On one hand, it facilitates navigation. But it can also cause the user to "get lost." Don't overdo horizontal connections within a hierarchy.*

*Hierarchical structures* (Figure 29.5) are undoubtedly the most common WebApp architecture. Unlike the partitioned software hierarchies discussed in Chapter 14 which encourage flow of control only along vertical branches of the hierarchy, a WebApp hierarchical structure can be designed in a manner that enables (via hypertext branching) flow of control horizontally, across vertical branches of the structure. Hence, content presented on the far left-hand branch of the hierarchy can have hypertext links that lead to content that exists in the middle or right-hand branch of the structure. It should be noted, however, that although such branching allows rapid navigation across WebApp content, it can lead to confusion on the part of the user.

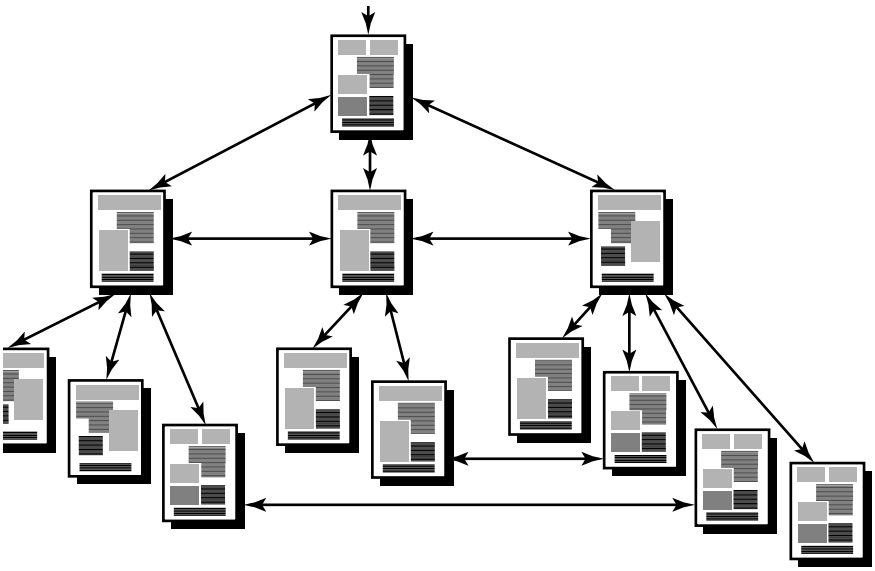


**FIGURE 29.5**
Hierarchical structures

A *networked,* or "pure Web," structure (Figure 29.6) is similar in may ways to the architecture that evolves for object-oriented systems. Architectural components (in this case, Web pages) are designed so that they may pass control (via hypertext links) to virtually every other component in the system. This approach allows considerable navigation flexibility, but at the same time, can be confusing to a user.

The architectural structures discussed in the preceding paragraphs can be combined to form composite structures. The overall architecture of a WebApp may be hierarchical, but one part of the structure may exhibit linear characteristics, while

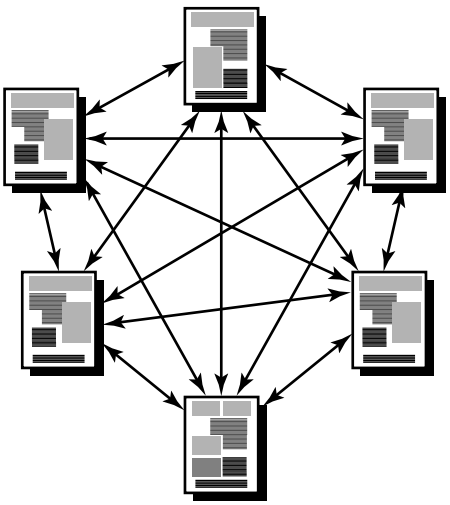


**FIGURE 29.6**
Networked, or "pure Web," structure

another part of the architecture may be networked. The goal for the architectural designer is to match the WebApp structure to the content to be presented and the processing to be conducted.

### Design Patterns

**XRef**

Further discussion of design patterns may be found in Chapters 14 and 22.

As we noted earlier in this book, design patterns are a generic approach for solving some small problem that can be adapted to a much wider variety of specific problems. In the context of Web-based systems and applications, design patterns can be applied at the architectural level, the component-level, and at the hypertext (navigational) level.

When data processing functionality is required within a WebApp, the architectural and component-level design patterns proposed by [BUS96], [GAM95], and others are applicable. Hypertext-level design patterns focus on the design of navigation features that allow a user to move through WebApp content in a facile manner. Among many hypertext design patterns proposed in the literature are [BER98]:

**Quote:**

"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution."

**Christopher Alexander**

- *Cycle*—a pattern that returns the user to a previously visited content node.
- *Web ring*—a pattern that implements a "grand cycle that links entire hypertexts in a tour of a subject" [BER98].
- *Contour*—a pattern that occurs when cycles impinge upon one another, allowing navigation across paths defined by the cycles.
- *Counterpoint*—a pattern that adds hypertext commentary, which interrupts content narrative to provide additional information or insight.
- *Mirrorworld*—content is presented using different narrative threads, each with a different point of view or perspective. For example, content that describes a personal computer might allow the user to select a "technical" or "nontechnical" narrative that describes the machine.
- *Sieve*—a pattern that guides a user through a series of options (decisions) in order to direct the user to specific content indicated by the sequence of options chosen or decisions made.
- *Neighborhood*—a pattern that overlays a uniform navigational frame across all Web pages in order to allow a user to have consistent navigation guidance regardless of location within the WebApp.

These hypertext design patterns can be reused as content is translated into a format that allows navigation through a WebApp.

## 29.5.2 Navigation Design

Once the WebApp architecture has been established and the components (pages, scripts, applets, and other processing functions) of the architecture have been identified, the designer must define navigation pathways that enable a user to access

WebApp content and services. To accomplish this, the designer must (1) identify the semantics of navigation for different users of the site and (2) define the mechanics (syntax) of achieving the navigation.

A large WebApp will often have a variety of different user roles. For example, roles might be *visitor, registered customer,* or *privileged user.* Each of these roles can be associated with different levels of content access and different services. A *visitor* may have access to only limited content while a *registered customer* may have access to a much broader range of information and services. The semantics of navigation for each of these two roles would be different.

The WebApp designer creates a *semantic navigation unit* (SNU) for each goal associated with each user role [GNA99]. For example, a *registered customer* may have six different goals, all resulting in access to different information and services. A SNU is created for each goal. Gnaho and Larcher [GNA99] describe the SNU in the following way:


A SNU is composed of a set of navigational substructures called *ways of navigating* (WoN). The SNU represents a specific navigational goal for a specific type of user.

> The structure of an SNU is composed of a set of navigational sub-structures that we call *ways of navigating* (WoN). A WoN represents the best navigation way or path for users with certain profiles to achieve their desired goal or sub-goal. Therefore, the concept of WoN is associated to the concept of User Profile.
>
> The structure of a WoN is made out of a set of relevant *navigational nodes* (NN) connected by *navigational links,* including sometimes other SNUs. That means that SNUs may themselves be aggregated to form a higher-level SNU, or may be nested to any depth.

During the initial stages of navigation design, the WebApp structure (architecture and components) is assessed to determine one or more WoN for each user goal. As noted, a WoN identifies navigation nodes (e.g., Web pages) and then links that enable navigation between them. The WoN are then organized into SNUs.

As design proceeds, the mechanics of each navigation link are identified. Among many possible options are text-based links, icons, buttons and switches, and graphical metaphors. The designer must choose navigation links that are appropriate for the content and consistent with the heuristics that lead to high-quality interface design.

In addition to choosing the mechanics of navigation, the designer should also establish appropriate navigation conventions and aids. For example, icons and graphical links should look "clickable" by beveling the edges to give the image a three-dimensional look. Audio or visual feedback should be designed to provide the user with an indication that a navigation option has been chosen. For text-based navigation, color should be used to indicate navigation links and to provide an indication of links already traveled. These are but a few of dozens of design conventions that make navigation user-friendly. In addition to conventions, navigation aids such as site maps, tables of contents, indexes, search mechanisms, and dynamic help facilities should also be designed at this time.

### 29.5.3 Interface Design

**Quote:**

"People have very little patience for poorly designed WWW sites."

**Jakob Nielsen and Annette Wagner**

The interface design concepts, principles, and methods presented in Chapter 15 are all applicable to the design of user interfaces for WebApps. However, the special characteristics of Web-based systems and applications require a number of additional considerations.

The user interface of a WebApp is its "first impression." Regardless of the value of its content, the sophistication of its processing capabilities and services, and the overall benefit of the WebApp itself, a poorly designed interface will disappoint the potential user and may, in fact, cause the user to go elsewhere. Because of the sheer volume of competing WebApps in virtually every subject area, the interface must "grab" a potential user immediately. Nielsen and Wagner [NIE96] suggest a few simple guidelines based on their redesign of a major WebApp:

**? What are some basic guidelines for the design of a Web site's look and feel?**

- Server errors, even minor ones, are likely to cause a user to leave the Web site and look elsewhere for information or services.
- Reading speed on a computer monitor is approximately 25 percent slower than reading speed for hard copy. 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 headbars 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.
- Navigation options should be obvious, even to the casual user. The user should not have to search the screen to determine how to link to other content or services.


One of the Web's best collections of usability resources has been assembled by The Usability Group and can be found at **www.usability.com/umi_links.htm**

A well-designed interface improves the user's perception of the content or services provided by the site. It need not be flashy, but it should always be well structured and ergonomically sound. A comprehensive discussion of WebApp user interfaces is beyond the scope of this book. Interested readers should see [SAN96], [SPO98], [ROS98], or [LYN99] among hundreds of offerings for additional guidance.

## 29.6 TESTING WEB-BASED APPLICATIONS

In Chapter 17, 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, hardware platforms, and communications protocols, the search for errors represents a significant challenge for Web engineers.

The approach for WebApp testing adopts the basic principles for all software testing (Chapter 17) and applies a strategy and tactics that have been recommended for object-oriented systems (Chapter 23). The following steps summarize the approach:

**What steps do we follow to test a WebApp?**

1. **The content model for the WebApp is reviewed to uncover errors.** This "testing" activity is similar in many respects to copy editing 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.
2. **The design model for the WebApp is reviewed to uncover navigation errors.** Use-cases, derived as part of the analysis activity, allow a Web engineer to exercise each usage scenario against the architectural and navigation design. In essence, these nonexecutable tests help uncover errors in navigation (e.g., a case where the user cannot reach a navigation node). In addition, the navigation links (Section 29.5.2) are reviewed to ensure that they correspond with those specified in each SNU for each user role.
3. **Selected processing components and Web pages are unit tested.** When WebApps are considered, the concept of the unit changes. Each Web page encapsulates content, navigation links, and processing elements (forms, scripts, applets). It is not always possible or practical to test each of these characteristics individually. In many cases, the smallest testable unit is the Web page. Unlike unit testing of conventional software, which tends to focus on the algorithmic detail of a module and the data that flow across the module interface, page-level testing for WebApps is driven by the content, processing, and links encapsulated by the Web page.
4. **The architecture is constructed and integration tests are conducted.** The strategy for integration testing depends on the architecture that has been chosen for the WebApp. If the WebApp 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 23) can be used to integrate the set of Web pages (a SNU may be used to define the appropriate set) required to respond to a user event. Each thread is

**Quote:**

"Innovation is a bittersweet deal for software testers. Just when it seems that we know how to test a particular technology, a new one [the Internet and WebApps] comes along and all bets are off."

**James Bach**

**XRef**

Integration strategies are discussed in Chapters 18 and 23.

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 the use-cases and SNU). Test cases are derived to uncover errors in the collaborations.

5. **The assembled WebApp is tested for overall functionality and content delivery.** Like conventional validation, the validation of Web-based systems and applications focuses on user-visible actions and user-recognizable output from the system. To assist in the derivation of validation tests, the tester should draw upon use-cases. The use-case provides a scenario that has a high likelihood of uncovering errors in user interaction requirements.
6. **The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration.** A cross-reference matrix that defines all probable operating systems, browsers,[7] hardware platforms, and communications protocols is created. Tests are then conducted to uncover errors associated with each possible configuration.
7. **The WebApp is tested by a controlled and monitored population of end-users.** A population of users that encompasses every possible user role is chosen. The WebApp is exercised by these users and the results of their interaction with the system are evaluated for content and navigation errors, usability concerns, compatibility concerns, and WebApp reliability and performance.

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

## 29.7 MANAGEMENT ISSUES

Given the immediacy of WebApps, it is reasonable to ask: "Do we really need to spend time managing a WebApp effort? Shouldn't we just let a WebApp evolve naturally, with little or no explicit management?" More than a few Web developers would opt for little or no management, but that doesn't make them right!

**XRef**

The activities associated with software project management are discussed in Part Two of this book.

Web engineering is a complicated technical activity. Many people are involved, often working in parallel. The combination of technical and nontechnical tasks that must occur (on time and within budget) to produce a high-quality WebApp represents a challenge for any group of professionals. In order to avoid confusion, frustration, and failure, planning must occur, risks must be considered, a schedule must be established and tracked, and controls must be defined. These are the core activities that we have called *project management.*

7 Browsers are notorious for implementing their own subtly different "standard" interpretations of HTML and Javascript. See www.browsercaps.com for a discussion of compatibility issues.

### 29.7.1 The WebE Team

**Quote:**

"In today's net-centric and Web-enabled world, one now needs to know a lot about a lot."

**Scott Tilly and Shihoug Huang**

The creation of a successful Web application demands a broad array of skills. Tilley and Huang [TIL99] address this issue when they state: "There are so many different aspects to [Web] application software that there is a (re)emergence of the renaissance person, one who is comfortable operating in several disciplines . . ." While the authors are absolutely correct, "renaissance" people are in relatively short supply; and given the demands associated with major WebApp development projects, the diverse skill set required might be better distributed over a WebE team.

WebE teams can be organized in much the same way as the software teams discussed in Chapter 3. However, the players and their roles are often quite different. Among the many skills that must be distributed across WebE team members are component-based software engineering, networking, architectural and navigational design, Internet standards/languages, human interface design, graphic design, content layout, and WebApp testing. The following roles[8] should be distributed among the members of the WebE team:

**? What roles do people play on a WebE team?**

**Content developer and providers.** Because WebApps are inherently content driven, one WebE team member role must focus on the generation or collection of content. Recalling that content spans a broad array of data objects, content developers and providers may come from diverse (non-software) backgrounds. For example, marketing or sales staff may provide product information and graphical images, media producers may provide video and audio, graphic designers may provide layout design and aesthetic content, copywriters may provide text-based content. In addition, research staff may be required to find and format external content for placement or reference within the WebApp.

**Web publisher.** The diverse content generated by content developers and providers must be organized for inclusion within the WebApp. In addition, someone must act as liaison between technical staff that engineers the WebApp and nontechnical content developers and providers. This role is filled by the Web publisher, who must understand both content and WebApp technology including HTML (or its next generation extensions, such as XML), database functionality, scripts, and general Web-site navigation.

**Web engineer.** A Web engineer becomes involved in a wide range of activities during the development of a WebApp including requirements elicitation; analysis modeling; architectural, navigational, and interface design; WebApp implementation; and testing. The Web engineer should also have a solid understanding of component technologies, client/server architectures, HTML/XML, and database technologies as well as a working knowledge of

8 These roles have been adapted from Hansen, Deshpande, and Murgusan [HAN99].

multi-media concepts, hardware/software platforms, network security, and Web-site support issues.

**Support specialist.** This role is assigned to the person (people) who has responsibility for continuing WebApp support. Because WebApps continuously evolve, the support specialist is responsible for corrections, adaptations, and enhancements to the site, including updates to content, implementation of new procedures and forms, and changes to the navigation pattern.

**Administrator.** Often called the *Web master,* this person has responsibility for the day-to-day operation of the WebApp, including

- Development and implementation of policies for the operation of the WebApp.
- Establishment of support and feedback procedures.
- Implementation of security procedures and access rights.
- Measurement and analysis of Web-site traffic.
- Coordination of change control procedures (Section 29.7.3).
- Coordination with support specialists.

The administrator may also be involved in the technical activities performed by Web engineers and support specialists.

### 29.7.2 Project Management

In Part Two of this book, we considered each of the activities that are collectively called *project management.*[9] Process and project metrics, project planning (and estimation), risk analysis and management, scheduling and tracking, SQA and SCM were all considered in some detail. In theory, most (if not all) of the the project management activities discussed in earlier chapters apply to WebE projects. But in practice, the WebE approach to project management is considerably different.

First, a substantial percentage[10] of WebApps are outsourced to vendors who (purportedly) specialize in the development of Web-based systems and applications. In such cases, a business (the customer) asks for a fixed price quote for WebApp development from two or more vendors, evaluates competing quotes, and then selects a vendor to do the work. But what does the contracting organization look for? How is the competence of a WebApp vendor determined? How does one know whether a price quote is reasonable? What degree of planning, scheduling, and risk assessment can be expected as an organization (and its outsourcing contractor) embarks on a major WebApp development effort?

---

9 Readers who are unfamiliar with basic project management concepts are urged to review Chapter 3 at this time.

10 Although reliable industry data are difficult to find, it is safe to say that this percentage is considerably higher than the one encountered in conventional software work.

Second, WebApp development is a relatively new application area and there is little historical data to use for estimation. To date, virtually no WebE metrics have been published in the literature. In fact, relatively little discussion has emerged on what those metrics might be. Therefore, estimation is purely qualitative—based on past experience with similar projects. But almost every WebApp wants to be innovative—offering something new and different to those that use it. Hence, experiential estimation, although useful, is open to considerable error. Therefore, how are reliable estimates derived? What degree of assurance can be given that defined schedules will be met?

Third, estimation, risk analysis, and scheduling are all predicated on a clear understanding of project scope. And yet, the "continuous evolution" characteristic discussed in Section 29.1 suggests that WebApp scope will be fluid. How can the contracting organization and the outsourcing vendor control costs and schedule when requirements are likely to change dramatically as a project progresses? How can scope creep be controlled, and more important, should it be controlled, given the unique nature of Web-based systems and applications?

At this stage in the history of project management for WebApps, the questions precipitated by the differences just noted are not easy to answer. However, a few guidelines are worth considering.

**Quote:**

"My advice to builders [of WebApps]?
1. State your quality requirements quantitatively.
2. Get contractual guarantees . . .
3. Use proven track record suppliers . . .
4. Build and expand the system in evolutionary stages.
5. Consider appropriate systematic redundancy at many levels to allow some degree of operation when failures occur."

**Tom Gilb**

**Initiating a project.** Even if outsourcing is the strategy to be chosen for WebApp development, an organization must perform a number of tasks before searching for an outsourcing vendor to do the work:

1. *Many of the analysis activities discussed in Section 29.3 should be performed internally.* The audience for the WebApp is identified; internal stakeholders who may have interest in the WebApp are listed; the overall goals for the WebApp are defined and reviewed; the information and services to be delivered by the WebApp are specified; competing Web sites are noted; and qualitative and quantitative "measures" of a successful WebApp are defined. This information should be documented in a product specification.
2. *A rough design for the WebApp should be developed internally*. Obviously, an expert Web developer will create a complete design, but time and cost can be saved if the general look and feel of the WebApp is identified for the outsourcing vendor (this can always be modified during preliminary stages of the project). The design should include an indication of the type and volume of content to be presented by the WebApp and the types of interactive processing (e.g., forms, order entry) to be performed. This information should be added to the product specification.
3. *A rough project schedule, including not only final delivery dates but also milestone dates, should be developed.* Milestones should be attached to deliverable versions of the WebApp as it evolves.

4. *The degree of oversight and interaction by the contractor with the vendor should be identified.* This should include the naming of a vendor liaison and the identification of the liaison's responsibilities and authority, the definition of quality review points as development proceeds, and the vendor's responsibilities with respect to interorganizational communication.

All of the information developed during these steps should be organized into a *request for quote* that is transmitted to candidate vendors.[11]

**Quote:**
"If you're only willing to pay peanuts, you get monkeys."
**from "The A Team"**

**Selection of candidate outsourcing vendors.** In recent years, thousands of "Web design" companies have emerged to help businesses establish a Web presence or engage in e-commerce. Many have become adept at the WebE process, but many others are little more than hackers. In order to select candidate Web developers, the contractor must perform due diligence: (1) interview past clients to determine the Web vendor's professionalism, ability to meet schedule and cost commitments, and ability to communicate effectively; (2) determine the name of the vendor's chief Web engineer(s) for successful past projects (and, later, be certain that this person is contractually obligated to be involved in your project); and (3) carefully examine samples of the vendor's work that are similar in look and feel (and business area) to the WebApp that is to be contracted. Even before a request for quote is offered, a face-to-face meeting may provide substantial insight into the "fit" between contractor and vendor.

**Assessing the validity of price quotes and the reliability of estimates.** Because relatively little historical data exist and the scope of WebApps is notoriously fluid, estimation is inherently risky. For this reason, some vendors will embed substantial safety margins into the cost quoted for a project. This is both understandable and appropriate. The question is *not* "Have we gotten the best bang for our buck?" Rather, the questions should be

- Does the quoted cost of the WebApp provide a direct or indirect return on investment that justifies the project?
- Does the vendor that has provided the quote exhibit the professionalism and experience we require?

If the answers to these questions are, "Yes," the price quote is fair.

**The degree of project management you can expect or perform.** The formality associated with project management tasks (performed by both the vendor and the contractor) is directly proportional to the size, cost, and complexity of the WebApp. For large, complex projects, a detailed project schedule that defines work tasks, SQA

11 If WebApp development work is to be conducted by an internal group, nothing changes! The project is initiated in the same manner.

checkpoints, engineering work products, customer review points, and major milestones should be developed. The vendor and contractor should assess risk jointly and develop plans for mitigating, monitoring, and managing those risks that are deemed important. Mechanisms for quality assurance and change control should be explicitly defined in writing. Methods for effective communication between the contractor and the vendor should be established.

**Assessing the development schedule.** Because WebApp development schedules span a relatively short period of time (often less than one or two months), the development schedule should have a high degree of granularity. That is, work tasks and minor milestones should be scheduled on a daily timeline. This fine granularity allows both the contractor and the vendor to recognize schedule slippage before it threatens the final completion date.

**Managing scope.** Because it is highly likely that scope will change as a WebApp project moves forward, the WebE process model should be incremental (Chapter 2). This allows the development team to "freeze" the scope for one increment so that an operational WebApp release can be created. The next increment may address scope changes suggested by a review of the preceding increment, but once the second increment commences, scope is again frozen temporarily. This approach enables the WebApp team to work without having to accommodate a continual stream of changes but still recognizes the continuous evolution characteristic of most WebApps.

These guidelines are not intended to be a foolproof cookbook for the production of low-cost, on-time WebApps. However, they will help both the contractor and the vendor initiate work smoothly with a minimum of misunderstandings.

### 29.7.3 SCM Issues for WebE

**Quote:**

"The most effective way to cope with change is to help create it."

**L. W. Lynett**

Over the past decade, WebApps have evolved from informal devices for information dissemination to sophisticated sites for e-commerce. As WebApps become increasingly important to business survival and growth, the need for configuration control grows. Why? Because, without effective controls, improper changes to a WebApp (recall that immediacy and continuous evolution are the dominant attributes of many WebApps) can lead to (1) unauthorized posting of new product information, (2) erroneous or poorly tested functionality that frustrates visitors to a Web site, (3) security holes that jeopardize internal company systems, and other economically unpleasant or even disastrous consequences.

The general strategies for software configuration management (SCM) described in Chapter 9 are applicable, but tactics and tools must be adapted to conform to the unique nature of WebApps. Four issues [DAR99] should be considered when developing tactics for WebApp configuration management—content, people, scalability, and politics.

**Content.** A typical WebApp contains a vast array of content—text, graphics, applets, scripts, audio and video files, forms, active page elements, tables, streaming data, and many others. The challenge is to organize this sea of content into a rational set of configuration objects (Chapter 9) and then establish appropriate configuration control mechanisms for these objects. One approach is to model the WebApp content using conventional data modeling techniques (Chapter 11), attaching a set of specialized properties to each object. The static or dynamic nature of each object and its projected longevity (e.g., temporary, fixed existence, or permanent object) are examples of properties that are required to establish an effective SCM approach. For example, if a content item is changed hourly, it has *temporary longevity.* The control mechanisms for this item would be different (less formal) than those applied for a forms component that is a permanent object.

**People.** Because a significant percentage of WebApp development continues to be conducted in an ad hoc manner, any person involved in the WebApp can (and often does) create content. Many content creators have no software engineering background and are completely unaware of the need for configuration management. The application grows and changes in an uncontrolled fashion.


*Controlling change during WebE projects is essential, but it can be overdone. Begin with relatively informal change control procedures (Chapter 9). Your initial goal should be to avoid the ratcheting effects of uncontrolled change.*

**Scalability.** The techniques and controls applied to small WebApps do not scale upward well. It is not uncommon for a simple WebApp to grow significantly as interconnections with existing information systems, databases, data warehouses, and portal gateways are implemented. As size and complexity grows, small changes can have far-reaching and unintended effects that can be problematic. Therefore, the rigor of configuration control mechanisms should be directly proportional to application scale.

**Politics.** Who "owns" a WebApp? This question is argued in companies large and small, and its answer has a significant impact on the management and control activities associated with WebE. In some instances Web developers are housed outside the IT organization, creating potential communication difficulties. Dart [DAR99] suggests the following questions to help understand the politics associated with WebE: Who assumes responsibility for the accuracy of the information on the Web site? Who ensures that quality control processes have been followed before information is published to the site? Who is responsible for making changes? Who assumes the cost of change? The answers to these questions help determine the people within an organization who must adopt a configuration management process for WebApps.

Configuration management for WebE is in its infancy. A conventional SCM process may be too cumbersome. The vast majority of SCM tools lack the features that allow

them to be adapted easily to WebE. Among the issues that remain to be addressed are [DAR99]

- How can we create a configuration management process that is nimble enough to accommodate the immediacy and continuous evolution of WebApps?
- How can we best introduce configuration management concepts and tools to developers who are completely unfamiliar with the technology?
- How can we provide support for distributed WebApp development teams?
- How can we provide control in a quasi-publishing environment where content changes on a nearly continuous basis?
- How can we attain the granularity required to control a large array of configuration objects?
- How can we incorporate configuration management functionality into existing WebE tools?
- How can we manage changes to objects that contain links to other objects?

These and many other issues must be addressed before effective configuration management is available for WebE.

## 29.8 SUMMARY

The impact of Web-based systems and applications is arguably the single most significant event in the history of computing. As WebApps grow in importance, a disciplined Web engineering approach—based on the principles, concepts, process, and methods that have been developed for software engineering—has begun to evolve.

WebApps are different from other categories of computer software. They are network intensive, content driven, and continuously evolving. The immediacy that drives their development, the overriding need for security in their operation, and the demand for aesthetic as well as functional content delivery are additional differentiating factors. Three technologies—component-based development, security, and Internet standard markup languages—are integrated with more conventional software engineering technologies during WebApp development.

The Web engineering process begins with formulation—an activity that identifies the goals and objectives of the WebApp. Planning estimates overall project cost, evaluates risks associated with the development effort, and defines a development schedule. Analysis establishes technical requirements for the WebApp and identifies the content items that will be incorporated. The engineering activity incorporates two parallel tasks: content design and technical design. Page generation is a construction activity that makes heavy use of automated tools for WebApp creation; and testing exercises WebApp navigation, attempting to uncover errors in function and content,

while ensuring that the WebApp will operate correctly in different environments. Web engineering makes use of an iterative, incremental process model because the development timeline for WebApps is very short. The umbrella activities applied during software engineering work—SQA, SCM, project management—apply to all Web engineering projects.

## REFERENCES

[ATK97] Atkins, D., et al., *Internet Security: Professional Reference,* New Riders Publishing, 2nd ed., 1997.

[BER98] Bernstein, M., "Patterns in Hypertext," *Proc. 9th ACM Conf. Hypertext,* ACM Press, 1998, pp. 21–29.

[BRA97] Bradley, N., *The Concise SGML Companion,* Addison-Wesley, 1997.

[BRE99] Brenton, C., *Mastering Network Security,* Sybex, 1999.

[BUS96] Buschmann, F., et al., *Pattern-Oriented Software Architecture,* Wiley, 1996.

[DAR99] Dart, S., "Containing the Web Crisis Using Configuration Management," *Proc. First ICSE Workshop on Web Engineering,* ACM, Los Angeles, May 1999. (The proceedings of the First ICSE Workshop on Web Engineering are published on-line at http://fistserv.macarthur.uws.edu.au/san/icse99-WebE/ICSE99-WebE-Proc/default.htm).

[DIN98] Dinucci, D., M. Giudice, and L. Stiles, *Elements of Web Design: The Designer's Guide to a New Medium,* 2nd ed., Peachpit Press, 1998.

[GAM95] Gamma, E., et al., *Design Patterns,* Addison-Wesley, 1995.

[GNA99] Gnaho, C. and F. Larcher, "A User Centered Methodology for Complex and Customizable Web Applications Engineering," *Proc. First ICSE Workshop on Web Engineering,* ACM, Los Angeles, May 1999.

[HAN99] Hansen, S., Y. Deshpande, and S. Murugesan, "A Skills Hierarchy for Web Information System Development," *Proc. First ICSE Workshop on Web Engineering,* ACM, Los Angeles, May 1999.

[ISA95] Isakowitz, T., et al., "RMM: A Methodology for Structured Hypermedia Design," *CACM,* vol. 38., no. 8, August 1995, pp. 34–44.

[KAE99] Kaeo, M., *Designing Network Security,* Cisco Press, 1999.

[LOW99] Lowe, D., "Web Engineering or Web Gardening?" *WebNet Journal,* vol. 1, no. 2, January–March 1999.

[LYN99] Lynch, P.J. and S. Horton, *Web Style Guide: Basic Design Principles for Creating Web Sites,* Yale University Press, 1999.

[MUR99] Murugesan, S., WebE home page, http://fistserv.macarthur.uws.edu.au/san/WebEHome, July 1999.

[NAN98] Nanard, M. and P. Kahn, "Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates," *Proc. Ninth ACM Conf. on Hypertext and Hypermedia,* ACM Press, 1998, pp. 11–20.

[NIE96] Nielsen, J. and A. Wagner, "User Interface Design for the WWW," *Proc. CHI '96 Conference on Human Factors in Computing Systems,* ACM Press, 1996, pp. 330–331.
[NOR99] Norton, K., "Applying Cross Functional Evolutionary Methodologies to Web Development," *Proc. First ICSE Workshop on Web Engineering,* ACM, Los Angeles, May 1999.
[OLS99] Olsina, L., et al., "Specifying Quality Characteristics and Attributes for Web Sites," *Proc. First ICSE Workshop on Web Engineering,* ACM, Los Angeles, May 1999.
[PAR99] Pardi, W.J., *XML in Action,* Microsoft Press, 1999.
[POW98] Powell, T.A., *Web Site Engineering,* Prentice-Hall, 1998.
[PRE98] Pressman, R.S. (moderator), "Can Internet-Based Applications Be Engineered?" *IEEE Software,* September 1998, pp. 104–110.
[ROS98] Rosenfeld, L. and P. Morville, *Information Architecture for the World Wide Web,* O'Reilly and Associates, 1998.
[SAN96] Sano, D., *Designing Large-Scale Web Sites: A Visual Design Methodology,* Wiley, 1996.
[SCH96] Schwabe, D., G. Rossi, and S. Barbosa, "Systematic Hypermedia Application Design with OOHDM," *Proc. Hypertext '96,* pp. 116–128.
[SPO98] Spool, J.M., et al., *Web Site Usability: A Designer's Guide,* Morgan Kaufmann, 1998.
[STL99] St. Laurent, S. and E. Cerami, *Building XML Applications,* McGraw-Hill, 1999.
[TIL99] Tilley, S. and S. Huang, "On the Emergence of the Renaissance Software Engineer," *Proc. First ICSE Workshop on Web Engineering,* ACM, Los Angeles, May 1999.

## PROBLEMS AND POINTS TO PONDER

**29.1.** Are there other generic attributes that differentiate WebApps for more conventional software applications? Try to name two or three.

**29.2.** How do you judge the "quality" of a Web site? Make a ranked list of ten quality attributes that you believe are most important.

**29.3.** Do a bit of research and write a two- or three-page paper that summarizes one of the three technologies noted in Section 29.1.2.

**29.4.** Using an actual Web site as an example, illustrate the different manifestations of WebApp "content."

**29.5.** Answer the three formulation questions for a Web site that you're familiar with. Develop a statement of scope for the Web site.

**29.6.** Develop a set of user profiles for SafeHomeInc.com or a Web site assigned by your instructor.

**29.7.** Develop a complete list of informational and applicative goals for SafeHomeInc.com or a Web site assigned by your instructor.

**29.8.** Produce a set of use-cases for SafeHomeInc.com or a Web site assigned by your instructor.

**29.9.** How does content analysis differ from interaction and functional analysis?

**29.10.** Conduct a content analysis for SafeHomeInc.com or a Web site assigned by your instructor.

**29.11.** Suggest three "golden rules" that would help guide the design of WebApps.

**29.12.** How does a WebApp design pattern differ from a template?

**29.13.** Select a Web site with which you are familiar and develop a reasonably complete architectural design for the site. What architectural structure did the site designers select?

**29.14.** Do a bit of research and write a two- or three-page paper that summarizes current work in WebApp design patterns.

**29.15.** Develop an architectural design for SafeHomeInc.com or a Web site assigned by your instructor.

**29.16.** Develop SNUs for SafeHomeInc.com or a Web site assigned by your instructor.

**29.17.** Using an actual Web site as an example, develop a critique of its user interface and make recommendations that would improve it.

**29.18.** Describe how project management for Web-based systems and applications differs from project management for conventional software. How is it similar?

## FURTHER READINGS AND INFORMATION SOURCES

Hundreds of books that discuss one or more Web engineering topics have been published in recent years, although relatively few address all aspects of Web engineering. Lowe and Hall (*Hypertext and the Web: An Engineering Approach,* Wiley, 1999) and Powell [POW98] provide reasonably complete coverage. Norris, West, and Watson (*Media Engineering: A Guide to Developing Information Products,* Wiley, 1997); Navarro and Khan (*Effective Web Design: Master the Essentials,* Sybex, 1998); Fleming and Koman (*Web Navigation: Designing the User Experience,* O'Reilly and Associates, 1998); and Sano [SAN96] also cover important aspects of the engineering process.

In addition to [LYN99] and [DIN98], the following books provide useful guidance for technical and nontechnical (content and aesthetic) aspects of WebApp design:

Baumgardt, M., *Creative Web Design: Tips and Tricks Step by Step,* Springer-Verlag, 1998.

Donnelly, D., et al., *Cutting Edge Web Design: The Next Generation,* Rockport Publishing, 1998.

Holzschlag, M.E., *Web by Design: The Complete Guide,* Sybex, 1998.

Niederst, J. and R. Koman, *Web Design in a Nutshell: A Desktop Quick Reference,* O'Reilly and Associates, 1998.

Nielsen, J., *Designing Web Usability,* New Riders Publishing, 2000.

Weinman, L. and R. Pirouz, *Click Here: Web Communication Design,* New Riders Publishing, 1997.

Menasce and Almeida (*Capacity Planning for Web Performance: Metrics, Models, and Methods,* Prentice-Hall, 1998) address the quantitative assessment of WebApp performance. Amoroso *(Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response,* Intrusion.Net Books, 1999) provides detailed guidance for those Web engineers who specialize in security issues. Umar (*Application (Re)Engineering: Building Web-Based Applications and Dealing with Legacies,* Prentice-Hall, 1997) discusses strategies for the transformation (reengineering) of legacy systems into Web-based applications. Mosley (*Client Server Software Testing on the Desk Top and the Web,* Prentice-Hall, 1999) has written one of the few books that address testing issues associated with WebApps.

Haggard *(Survival Guide to Web Site Development,* Microsoft Press, 1998) and Siegel (*Secrets of Successful Web Sites: Project Management on the World Wide Web,* Hayden Books, 1997) provide guidance to managers who must control and track WebApp development.

A wide variety of information sources on 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/engcs/compsci/pressman/resources/webe.mhtml**

CHAPTER

# 30 REENGINEERING

KEY CONCEPTS



In a seminal article written for the *Harvard Business Review,* Michael Hammer [HAM90] laid the foundation for a revolution in management thinking about business processes and computing:

> It is time to stop paving the cow paths. Instead of embedding outdated processes in silicon and software, we should obliterate them and start over. We should "reengineer" our businesses: use the power of modern information technology to radically redesign our business processes in order to achieve dramatic improvements in their performance.
>
> Every company operates according to a great many unarticulated rules . . . Reengineering strives to break away from the old rules about how we organize and conduct our business.

Like all revolutions, Hammer's call to arms resulted in both positive and negative changes. During the 1990s, some companies made a legitimate effort to reengineer, and the results led to improved competitiveness. Others relied solely on downsizing and outsourcing (instead of reengineering) to improve their bottom line. "Mean" organizations with little potential for future growth often resulted [DEM95].

During this first decade of the twenty-first century, the hype associated with reengineering has waned, but the process itself continues in companies large

## QUICK LOOK

**What is it?** Consider any technology product that has served you well. You use it regularly, but it's getting old. It breaks too often, takes longer to repair than you'd like, and no longer represents the newest technology. What to do? If the product is hardware, you'll likely throw it away and buy a newer model. But if it's custom-built software, that option may not be available. You'll need to rebuild it. You'll create a product with added functionality, better performance and reliability, and improved maintainability. That's what we call reengineering.

**Who does it?** At a business level, reengineering is performed by business specialists (often consulting companies). At the software level, reengineering is performed by software engineers.

**Why is it important?** We live in a rapidly changing world. The demands on business functions and the information technology that supports them are changing at a pace that puts enormous competitive pressure on every commercial organization. Both the business and the software that supports (or is) the business must be reengineered to keep pace.

**What are the steps?** Business process reengineering (BPR) defines business goals, identifies and evaluates existing business processes, and creates revised business processes that better meet current goals. The software reengineering process