

Application Development

Chapter Outline

- Introduction, 396
- Two-Minute Chapter, 397
- Design Consistency, 398
 - Page Design Templates*, 398
 - Usability*, 399
 - Fonts and Customization*, 400
 - Mobile Devices*, 401
- Application Structure, 402
 - Designing Applications*, 403
 - The Startup Form*, 403
 - Sally's Pet Store: Application Organization*, 404
 - Administrative Tasks*, 407
- Menus and Toolbars, 407
 - Purpose of the Menu*, 408
 - Toolbars*, 409
 - Creating Menus and Toolbars*, 409
- Accessibility, 410
- Custom Help, 412
 - Creating a Help File for Windows*, 413
 - Context-Sensitive Help*, 415
 - Windows Help 3/Help Viewer*, 417
- Handling Errors, 419
 - Catching Errors*, 419
 - Logging Errors*, 420
 - Debugging*, 420
- Testing, 420
 - Form and Module Testing*, 421
 - Integrated Application Testing*, 422
 - Stress or Performance Testing*, 422
 - Usability Testing*, 422
 - Security Testing*, 423
- Deploying an Application, 424
 - Packaging Files*, 424
 - Installation Programs*, 425
 - Server and Database Configuration*, 425
- Summary, 425
- Key Terms, 426
- Review Questions, 427
- Exercises, 427
- Web Site References, 429
- Additional Reading, 429

What You Will Learn in This Chapter

- What features need to be included in finished applications?
- How do you create a consistent application design?
- How are forms and reports integrated and organized?
- How can users gain easy access to standard operations across the application?
- How can a computer application be modified for people with disabilities?
- How do you create custom help files?
- What does your application do when something goes wrong?
- How do you know your application works correctly?
- How will your application be installed?

A Developer's View

Miranda: Finally. I think I see the end of this project.

Ariel: That's terrific. What's left?

Miranda: Well, everyone is happy with the forms and reports. All I have to do now is tie them together into an application. I have a few details to add to make the forms a little easier to use. The salespeople complained about having to enter customer numbers twice, and they say the order lists are too long. They want to pick from a list of orders just for the given customer.

Ariel: That's it? Let's celebrate!

Miranda: Well, not quite yet. I also have to write some help files. Then I have to create a set of installation disks so they can install the system on all the computers.

Ariel: Sounds like a lot of details. Will it take long?

Miranda: I don't think so. But it will make the application more attractive and easier to use, so I really need to finish the details.

Getting Started

Applications are more than just forms and reports. You need to ensure all forms and reports have the same look and feel. You build an integrated application by linking everything with menus. You also need to add help files, add error handling, and test the entire application. Part of the testing includes ensuring that the application is accessible to all potential users. You also have to build administrative tools and deploy the application.

Introduction

What features need to be included in finished applications? As a database developer, it is your responsibility to create systems that help users in their jobs. You accomplish this task by building an **application** to perform a specific task. The task is defined by the user, and your application needs to be easy to use. The goal of the application is to collect data and provide information to help users make decisions. You define tables to hold the data, but you never want users to see the underlying tables. Instead, you create forms to collect data and reports to help users visualize and analyze the data.

An application is more than a collection of forms and reports. It is a set of forms and reports that work together. More specifically, an application has (1) an internal consistency to the user interface, (2) a structure or layout that supports the flow of user tasks, (3) menus to make it easy to find things, (4) a help system to provide documentation and assistance, (5) error handling to catch anticipated problems and protect the user, and (6) a method to deploy the application. At this stage in the design, you also have to perform considerable testing and evaluation of the application.

The first two items (consistency and structure) are the defining elements of an application. The others are features that are added at this stage to make the system more reliable and easier to use. The look and feel of the forms is the major

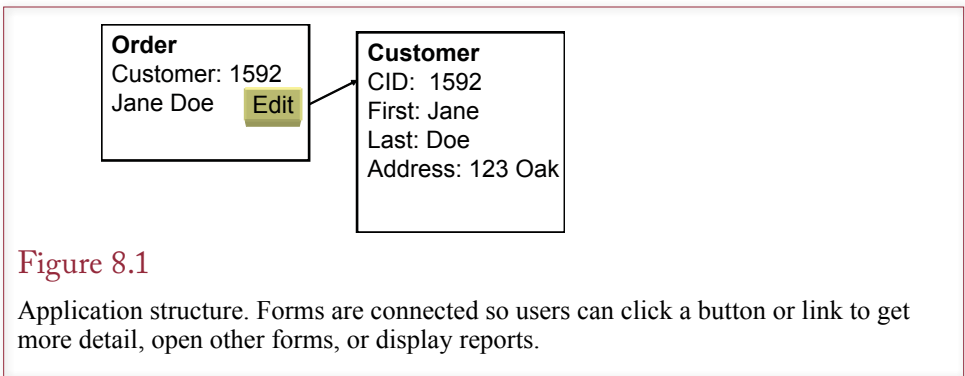


Figure 8.1

Application structure. Forms are connected so users can click a button or link to get more detail, open other forms, or display reports.

element of consistency and design. Every form in an application should have the same look and same approach to entering data. Otherwise users will get confused and become frustrated because your application makes it harder to perform their jobs. The topic of structure entails connecting forms and reports together. Users entering data on one form should be able to click a button, or double-click an entry to see more detail. Figure 8.1 shows a simple example. Users entering data on the Order form will probably want to open the Customer form to edit the data about the customer. As a developer, you have to learn what connections are needed on each form. These connections help create the structure of the application.

Two-Minute Chapter

Applications begin with forms and reports but require consistency, structure, and supporting details including menus, custom help topics, and the ability to handle errors. Unless you have been exceedingly careful from the start, be prepared to spend time to rebuild all forms and reports to ensure they have the same “look and feel.”

Applications need to have consistent colors, fonts, layouts, menus, and usability features. When possible, build templates that serve as the foundation for forms and reports. These templates usually set the page structure as well as the styles to be used in each component. Applications might need different templates and styles for different devices—particularly if mobile phones are going to be used with smaller screens. To improve accessibility, try to use system fonts and colors to support user control. Whenever possible, support multiple input methods, including keyboards.

The structure of an application is important—particularly where components are stored and run, as well as how pieces connect together. Most applications require some type of menu or toolbar along with buttons and links to help connect forms and reports and make it easier to find various elements and explore the application capabilities.

Applications also need custom help pages to answer user questions—particularly by providing the ability to search for keywords. Help topics are usually written as individual HTML topic pages. Keywords can be added to those pages along with using heading levels to indicate the table of contents or structure. Windows help files are created by compiling the HTML files into a single CHM file using the HTML help compiler from Microsoft or (expensive) third-party compilers. Context-sensitive help is provided by assigning a number to every page using the topics.h file, then entering the help file name and topic number into the application properties; such as the form properties in Microsoft Access.

Applications also need extensive testing, including module, integration, stress/performance, usability, and security testing. Special attention should be paid to check for SQL Injection attacks. At a minimum, all user inputs should be cleaned and SQL queries should use parameters instead of string concatenation. Even with extensive testing, errors can still arise, so the application should have error handling code that can automatically recover from errors whenever possible. It should also have the ability to log errors so developers can examine the log to look for common issues and find improvements for the next versions.

Applications also need a deployment method. Server-based systems are straightforward and might be deployed using basic script files. Client-based applications should be packaged and installable from a simple start command.

Design Consistency

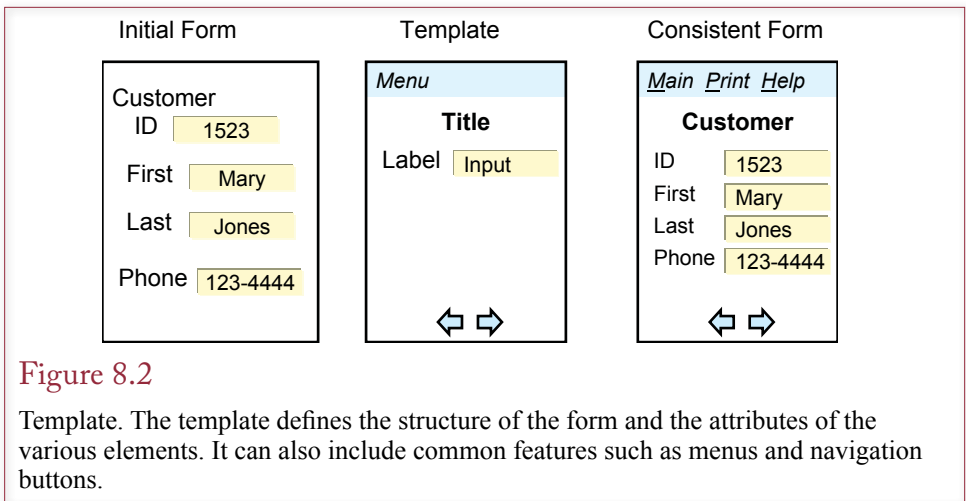
How do you create a consistent application design? This question is particularly important to answer when several developers are working on the same project. The application design consists of several levels. Some are easier to configure and observe. At the most basic level, all forms should use the same color scheme. Likewise, reports should use the same fonts and follow a similar layout. But, design also includes usability issues such as the page layout and selecting items from a list instead of memorizing ID numbers. Consistency in design also applies to the links between forms and the use of menus and toolbars. The primary issues discussed in this section are related to design and usability.

Page Design Templates

Look through a book, magazine, or well-designed application or Web site and you will see that all of the pages have a consistent appearance. Only a few fonts are used, headings are aligned the same, margins match, and colors blend and match across all of the pages. This consistency does not happen by accident. Graphics designers first create a template and then apply the template to all of the pages. A **template** defines the style features of a page. The capabilities depend on the system and tools available, but you can usually define the overall page layout, fonts, and colors.

Figure 8.2 illustrates the effect of a form template. Take a look through the forms you have created for your assignments and projects to this point. Do they look like the first form? When you look at a collection, do they all look the same, or are they all different? Eventually, developers learn to pay attention to detail and consistency. Even so, it is easy to make mistakes. A template makes it easier for everyone to be consistent and to reduce errors. Ultimately, someone still has to examine every form and report to double-check consistency, but with templates, you can come very close on the first pass.

Each development tool has different methods to create and apply templates. A few systems do not support templates at all, and some have predefined templates that you cannot change. The most common approach to templates is to apply the template at design time. If the template is changed later, there is no way to push the changes onto all of the forms based on the template. In other words: Be sure your template is complete and accurate before you create the forms and reports. If you have already created complex forms, it is difficult to apply a template. In some cases, you can create a new blank form based on the template and copy the controls from the original form. In extreme cases, you might have to rebuild the entire form from scratch if you need to apply a template later.



Application design is something that needs to be established early in the development process. Even if your system does not support templates, you can define a **style sheet** that defines the overall page layout, the common elements to include, and the styles of each major element (titles, labels, text, and so on). Each developer is responsible for following the guidelines on the style sheet. This process is harder to use than a template, but it works for every system.

Templates are not the final answer to design questions. For instance, some forms or reports may need to add other features, or change margins to make something fit on a page. Fortunately, once the template is applied, you can override a setting and coerce the page to get the layout you want. However, you need to be careful with this power. As a developer, you have to make decisions. It is best to maintain consistency, but ultimately, you have to keep the users happy. If a manager insists on squeezing an extra column onto a report, you will have to reduce the margins or change the font size. On large projects, you should establish a coordinator who can be consulted when you need to override a template specification.

Usability

Beyond layout, fonts, and colors, you need to establish a consistent set of usability standards for an application. For instance, consider the issue of foreign keys such as using CustomerID in the SalesOrder table. When a clerk is entering data into the SalesOrder table, it would be painful to require the clerk to memorize the CustomerID values. Instead, the form will have some type of drop-down list or list-of-values option that enables the clerk to pick the appropriate customer from a list. In terms of your application, you need to be sure that all foreign key references use the same approach to solve the problem. Consider the situation shown in Figure 8.3, where the Sales form uses a drop-down list to pick customers based on phone numbers. Imagine how annoying it would be to use a pop-up box to search for customers by name on the Receipts form.

When the same users are going to work with multiple forms, the forms need to use a consistent data-entry method. In general it is better to be consistent across all of the forms. In a few cases, one group of users might insist on a unique lookup approach that more closely matches its needs, but these variations should be discussed and approved separately.

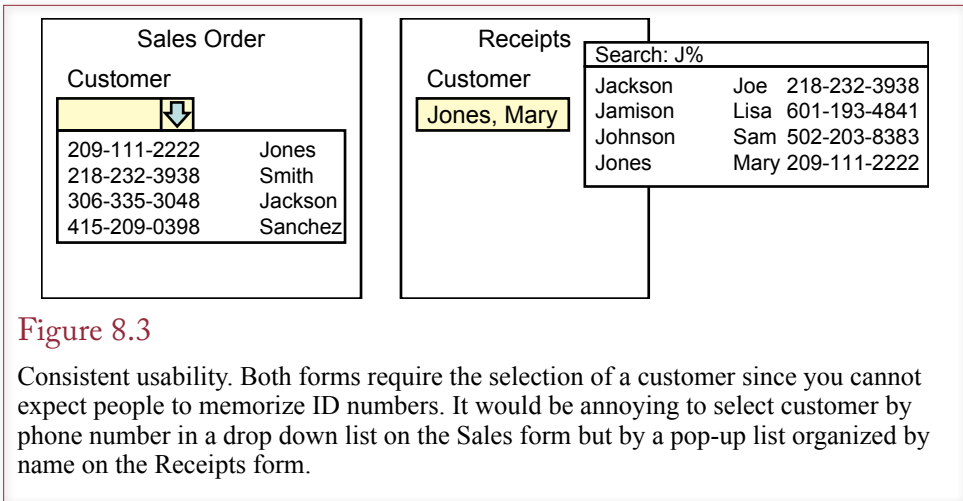


Figure 8.3

Consistent usability. Both forms require the selection of a customer since you cannot expect people to memorize ID numbers. It would be annoying to select customer by phone number in a drop down list on the Sales form but by a pop-up list organized by name on the Receipts form.

The idea of consistency also applies to the tab order, choice of related data to display on a form, subform layout, and similar topics. For example, when the user selects a customer, you might choose to display additional information, such as phone number, on the form. As much as possible, this same data should be displayed on all forms involving customer lookups.

Fonts and Customization

Selecting fonts and color schemes for an application is always a challenge. You face conflicting goals. On the one hand, you want to choose a pleasing design that displays all of the relevant information in one place. On the other hand, you need to give users control over the displays. Why do users need control? Users often configure their systems to support the way they work or to compensate for vision issues.

Users can configure the Windows environment through various settings in the Control Panel—including font sizes, color schemes, and regional settings such as date displays. Your application needs to support these settings. You provide this support by choosing system-defined fonts and colors. Windows development tools, such as Visual Studio, provide font settings for choosing system fonts. Use these choices instead of picking a fixed typeface and font size. Likewise, you should choose the Windows palette colors instead of forcing a fixed color. When your application runs, it will pick up the currently-defined fonts and colors. If the user changes those values, your application will adapt and use those colors. Yes, in some cases, the user might pick strange color combinations, but the decision belongs to the user—not to you.

Web applications are somewhat trickier—and the capabilities are heavily dependent on the development tools you use. At the moment, most systems do not provide user control over color schemes. Some default schemes are commonly used (e.g., white background, black text, and blue highlights), and you should follow these schemes when possible. On the other hand, font sizes are more flexible, and more interesting. Take a look at your browser options and you will find an option to control the font size (try Page/Text Size). However, if your Web form specifies fonts in terms of points, this option will not work for the users. Also, if you specify page layout sizes (e.g., tables) in terms of fixed measures such as pix-

Device	Size (Diagonal in.)	Pixels
Desktop	24	1920 x 1200
Laptop	15	1440 x 800 1920 x 1080
Apple iPad	11	2048 x 1536
Google Nexus 10	10	2560 x 1600
Apple iPhone 3S	3.5	480 x 320
Apple iPhone 5	4	1136 x 640
Samsung S4	5	1920 x 1080

Figure 8.4

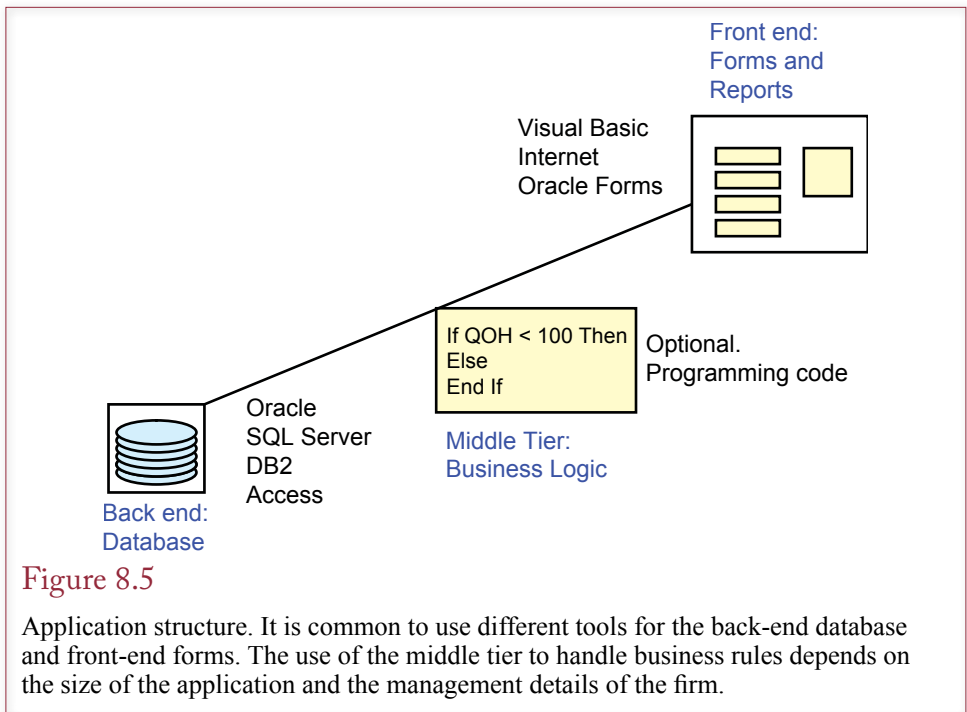
Sample device resolution. Fewer pixels means less information can be displayed. Smaller size means text and images might be too small to read, or less information can be displayed at readable fonts sizes.

els, changes in text size will not work very well as the font changes but the container remains the same size. Consequently, you have to use font and size settings based on relative terms instead of absolute point values. A relatively new way to define sizes is to use ems, where one em is defined as the width of the letter M in the current font. With relative sizes, the font and layout values will be rescaled automatically when the user changes the font size.

Mobile Devices

Increasingly, users want to access data from anywhere—using mobile devices connected to internal applications or Web sites. Providing users with greater access to applications and data is good. However, cell phones and other portable devices generally have smaller screens. Many wireless plans also have limitations on the data transfer speed or monthly caps (or high prices) on the amount of data transfer. Figure 8.4 shows approximate sizes and pixel counts for a handful of devices. Keep in mind that technology continues to evolve, so you need to research current values when you build an application. The main key is that the cell phones (Apple and Samsung) are considerably smaller and use fewer pixels. Although starting with the Apple iPhone 4 released in 2010, resolutions have improved—no one will be able to read the text if it is drawn pixel-for-pixel. Instead, the newer phones and tablets use those pixels to display better-formed characters—shown at readable size but with better resolution and clarity. Compare an iPhone screen at 3 x 2 inches with a desktop monitor at about 13 x 8 inches. Physically, the desktop screen can hold 16 iPhone screens. The pixel count works to about the same ratio with the 3S, but not the version 5. The point is that even if your application uses only half of the desktop monitor, only a small fraction of that page could be displayed on a mobile screen at one time.

Pages that work on larger screens can become unusable at smaller sizes—either the fonts are too small to read, or the user has to scroll vertically and horizontally to see the entire form. Unfortunately, there is no good answer to creating pages for different sizes of screens. Often, it is necessary to create two versions of the application—one for “regular” screens and one for smaller mobile screens. Creating different versions is also useful because mobile devices often have slower (or



more expensive) network connections, so you have to be more cautious in using graphics and limiting the amount of data sent to the page. But developing two versions of an application takes more time—hopefully less than twice as much—because you already have the tables, functions, and calculations. And you have to plan for more people to handle maintenance and upgrades.

Application Structure

How are forms and reports integrated and organized? The overall structure is an important feature of any application. The structure or layout defines how the user will deal with the application. Most database applications will use forms and reports as individual components. The first step in designing the structure or architecture of the application is to design each form. The objective of application structure is to organize all of the forms and reports to produce a complete application. In some applications, this purpose can be achieved with a central startup form, which contains buttons to direct users to the appropriate form. More commonly, you will also need to add interconnection buttons on individual forms. For example, a user entering data on an order form might want to look up additional information on the customer form.

Today, it is common to separate a database application into two or three major sections. As shown in Figure 8.5, the **front end** consists of the forms and reports that the user sees. The **back end** consists of the database tables. Sometimes a **middle tier** is added that consists of program code to define and enforce business rules. With a network, this separation can be physical, and each component can run on separate computers. Even if all of the elements will run on a single machine, it often makes sense to separate them logically. This separation enables you to change each part independently. It also makes it easier to choose different tools

for each purpose. For example, you could write the front end forms and reports with Visual Studio, store the data in an Oracle database, and write custom code to evaluate business rules. If the business changes, you could transfer the business rules to an enterprise resource planning system, or change the back-end DBMS with only minor changes on the front end.

Choosing the overall architecture of the application is the first step to designing the application. The choice of tools and structure will depend on the organization's needs and capabilities. Some companies have a formal process for designing and approving applications. This process is important when applications need to work together and fit into the overall structure of the company's information system.

Designing Applications

The first step in designing the application structure is to identify the various users and outline the tasks that will be performed with the application. The application must reflect the needs and working habits of the user. If several users have different needs, the application can be divided into sections for each group. A central startup form can be used to identify the user and direct him or her to the appropriate section.

This segmentation reduces complexity for the users and simplifies their tasks. However, it has two potential drawbacks. First, if the application has too many layers, users will have trouble finding the forms and reports they need. Second, poor organization confuses users and requires additional support and training. In other words, you must find an application structure that provides the functionality each user needs but is still easy to understand. The inherent conflict in these goals is what makes it so difficult to design a good application structure.

Even experienced programmers rarely design a "perfect" application the first time. In most cases you need to develop several ideas and test them. You can build **prototypes** by creating sample forms and including command buttons to tie them together. These prototypes can be given to users to test. You then incorporate user suggestions and modify the prototypes. By testing different structures, you can quickly learn which technique will work best.

In building a complex application structure, it is best to start with the core concepts. Once you have tested them with users, you can add features. Each revision constitutes a new application version. Keep track of the version number; record the date, the reason for the change, and the changes that were made. Most commercial software vendors follow this development process. No one tries to visualize a complete, massive application and create it up front. Instead, developers start with a basic concept and build a system that works and implements the fundamental concepts. Then developers expand the capabilities by adding new features.

The two most important aspects in this type of development are (1) getting the overall structure correct up front and (2) using a flexible design that is easy to modify later. For example, it is critical that your data tables be normalized, because normalized tables can be easily expanded later to provide new features.

The Startup Form

Designing an overall structure and appearance often requires artistic sensibility as well as logic and research. Each application is different and can require a unique approach. Yet, over time, designers have learned that some common elements can be used in many applications. The main menu or **startup form** is an element that many developers like to use. However, it is not really the ultimate answer to every problem.

- Directory for the application.
- Identify users.
- Startup and shutdown code.
 - Preload forms in background.
 - Initiate transaction and security logs.
 - Establish network connections.
- Copyright and usage notes.

Figure 8.6

Uses of startup forms. As the initial form for an application, the main menu can be used to control tasks that apply to the complete session.

The main purposes of the startup form are shown in Figure 8.6. The startup form is generally the first form of the application. It provides a centralized directory to the rest of the application. It often contains an image or picture and usually consists entirely of command buttons. Clicking a button brings the user to another menu form or to a specific form or report.

Because the startup form is the starting point for the application, it is a good place to identify the user. If possible, you should identify the user from the network login data. Otherwise, you will have to maintain a separate login for each user. The two primary reasons for identifying each user are (1) to maintain appropriate security controls and (2) to customize the application for each user group.

The startup form can be customized through layout and the use of color. For example, options primarily intended for different managers (marketing, finance, etc.) can be displayed in different colors. If additional customization is needed, individual options can be made invisible and disabled so that users see only buttons that are designed for their use. This approach simplifies the screen layout and reduces confusion. However, it is less useful if managers need to share their tasks.

Keep in mind that some applications will work better without a startup form. Think about the applications you use on a daily basis: word processing, spreadsheets, and so on. None of these require startup forms. Instead, they jump right to the primary user task and rely on menus to provide access to the functions. You could use this same approach for users who have a limited view of the application—such as front line clerks. Once a clerk logs in, the application jumps immediately to the primary data-entry form. The point is that you should not try to force a startup form into every application. Look at the jobs and talk with the users to find the best way to organize the forms and reports for each task.

Sally's Pet Store: Application Organization

In many ways the startup form is a table of contents into the application. It presents the organization of the application. Before building the startup form, you must decide how the application will be organized. That is, you must learn which forms are most important to users, how they will switch between forms, and how often they use each form.

Although there are many useful ways to organize any application, consider two different approaches to the Pet Store application. The first approach is shown in Figure 8.7. At first glance this approach seems reasonable. Items are ordered, then received and then sold to customers. Hence the store managers might want to start with orders and enter data by following each item from purchase through sale.

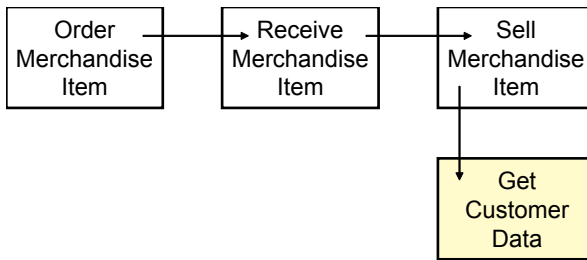


Figure 8.7

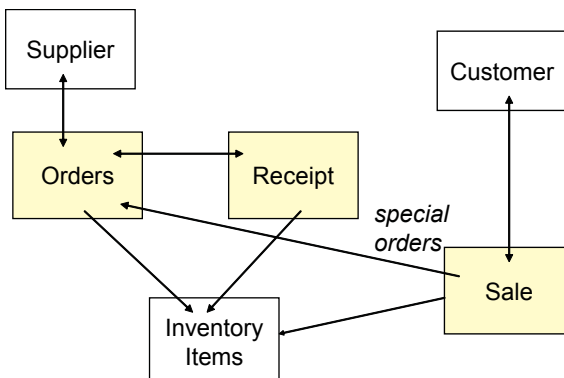
Poor organization of the Pet Store application. The links are at the wrong level (item instead of order). Managers rarely need to track individual items from order to receipt to sale.

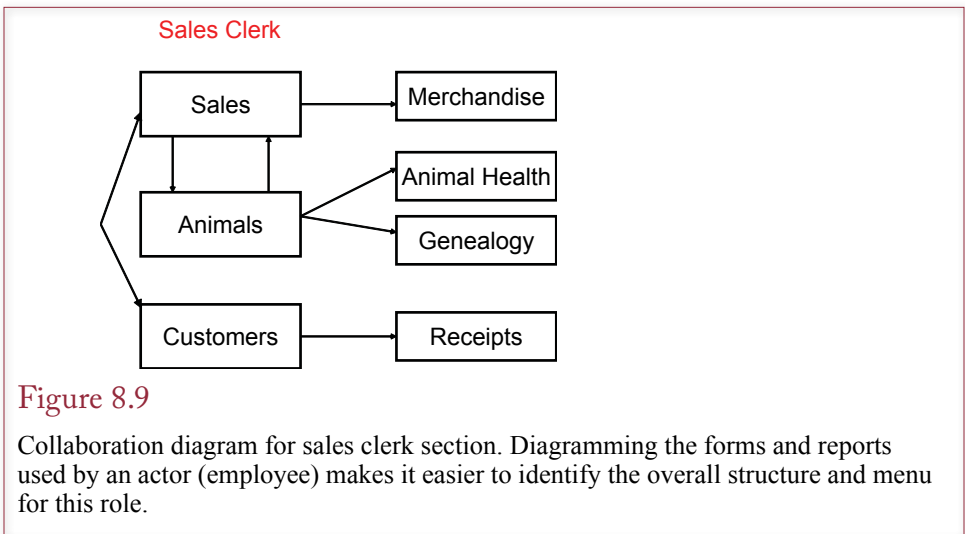
Although this approach might sound reasonable at first, it has several flaws. First, managers rarely want to track individual items. Perhaps they want to follow individual animals, but rarely merchandise. Second, when an order is placed, the item has not been received yet, so there is no point in linking an order to the receipt of the shipment. More important, there is no way to connect individual items to a sale. For example, you might know that a customer bought three cans of a particular dog food, but there is no way to tell exactly which cans. Hence, managers rarely need a link from receipt of shipments to individual sales.

An improved approach appears in Figure 8.8. First, notice that it has more links—including bidirectional links. For example, when a shipment arrives, workers need to pull up the matching order to see whether the proper items were delivered. Hence an Orders button is placed on the Shipping Receipt form. Once in a while, a manager might want to check on the shipment of a particular order, so the link is bidirectional. Notice that Sales are connected to Orders and Receipts—but only through the Inventory items. Inventory QOH can be displayed directly on the Sale form. The Sale form also has a connection to Orders—to create special

Figure 8.8

Improved organization for the Pet Store. The lines represent links from one form to a second form. The links are usually created through buttons placed on the form.





orders. If an item is out of stock, a salesperson might want to check on recent orders to see when the item might arrive. The designer should talk with users to determine how often this situation arises and how it should be handled on the form.

Eventually the Pet Store application will contain many forms and reports. Most of them are linked to a startup form. Many of them are linked to each other. Buttons or events on one form lead the user to a related form. Some of the forms are simple and affect one table, but most display data from several related tables. Each individual form represents specific business events and tasks. Figure 8.9 shows the primary forms used by the sales clerk role. This diagram is a simplified version of a UML collaboration diagram. The main point here is that it identifies the initial forms needed by this group of users. Consequently, you should put links to Sales, Animals, and Customers on the startup for this user. You will need to create a similar diagram for the other roles at the Pet Store, such as purchasing and management.

Figure 8.10 shows one version of a startup form for Sally's Pet Store. The buttons on this form match the primary tasks identified for the groups of users. The buttons are color coded to highlight the three groups. You could go further and set the visibility of the buttons based on the category of the user. When each person logs in, the form displays the buttons or forms most relevant to that person's tasks. With some additional coding, you could write the form so each user could select a set of buttons to personalize the main screen. However, customization is usually easier on tool bars instead of the main menu. In general, you should avoid putting too many buttons on the main menu. An old rule of thumb states that the average person can handle seven, plus or minus three, items at a time, so four to ten buttons on a form is a good target. Obviously, a complex application ultimately has many more than 10 forms and reports. With large applications, you can extend the startup form to additional forms or submenus. You can also add drop-down menus to make it easier to find commonly used items.

Remember that you also need to connect forms to other forms. Depending on the user interface you choose, you might add buttons to forms or use double-clicks to trigger the second form to open. These links are commonly created with foreign key relationships, such as adding an Edit button to the SalesOrder form to open



Figure 8.10

Sample startup form. The buttons match the user's tasks.

the Customer form. Similarly, you can add links to print or preview reports from various forms, such as printing a sales receipt from the SalesOrder form. The specific links depend on the needs of the users.

Administrative Tasks

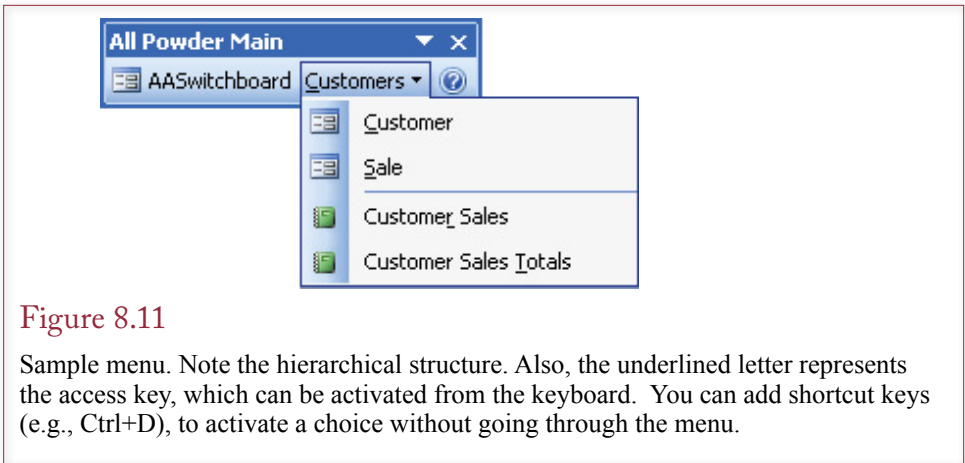
When you build the application, you also need to think about the administrative tasks that will need to be performed. **Administrative tasks** consist of jobs that need to be performed to keep the application running, such as updating data in lookup tables, backing up the database, and assigning users to groups. Depending on the DBMS, some of these tasks are handled outside of your application. If your system needs routine maintenance or tasks performed on a specified schedule, you should incorporate a set of administrative forms to automate the tasks. If nothing else, collecting the tasks into one location makes them easier to handle and increases the probability that they will actually be done. Tasks that require external steps could at least be documented within the application. Ultimately, you can hide the administrative tasks from the common users and use the security system to make them accessible to a few administrators.

Administrative forms are particularly important for Web-based applications. You will find that it is convenient to handle administrative tasks using a Web browser so the administrators can support the application from almost any location. This step is particularly critical when the application will be hosted by an external Internet service provider. On the other hand, it sometimes takes more work to create the administrative pages than it does to build the original application.

Menus and Toolbars

How can users gain easy access to standard operations across the application?

Contemporary applications have several features that are designed to standardize the look and feel of applications and to make your applications relatively easy to use. **Menus** or **toolbars** and the **Help system** are common elements in most applications. Menus and toolbars are similar to each other and often created using the same techniques. A toolbar is a collection of items that perform some action



when clicked. The items can be icons or text. Text items can be opened to provide drop-down lists of additional items. This submenu makes it easier to organize the many choices. A toolbar that primarily consists of text items is often referred to as a menu, but the distinction is minor. Most systems enable you to create multiple toolbars or menus. Generally, you can modify them on the fly in response to user actions. With some systems, it is relatively easy to enable the toolbars so users can configure their own icons and selections on a custom toolbar.

A main menu is generally the same across the application. Hence, the menu centralizes choices that can be activated at any time. Menus are also useful for visually challenged workers and those who prefer to use the keyboard instead of a pointing device (mouse), because choices can be activated with the keyboard. Toolbars usually consist of a set of icons or buttons that perform common tasks. Some applications enable users to customize the toolbar with specific buttons and users often reposition toolbars.

Most menus are hierarchical: that is, detailed choices are presented under a few keywords. The Windows interface standard specifies that menus should be displayed at the top of the application. However, users may want to move menus to a different location. Most applications use similar commands on their menus. For example, the menu in Figure 8.11 contains top-level links for the main startup form, customer information, and help. Ultimately, entries would be added to cover the other main objects such as suppliers and animals. Whenever you create text items on a menu, you should define an access key so that users can select the entry directly from the keyboard. In a Windows environment, items are activated with the Alt key combination, such as Alt+C to open the main customers menu.

Purpose of the Menu

You might consider using the basic DBMS menu within your application. Then users will have full control over the database. In most cases, however, you will be better off building a custom menu for your application. A custom menu has several benefits. First, it can limit user actions. For instance, if users do not need to delete data, the menu should not have the delete commands. You still have to set the appropriate security conditions to prevent them from using other methods to delete data, but removing a command from the menu helps to restrict user choices. A second advantage of a custom menu is that it simplifies the user interface. If en-

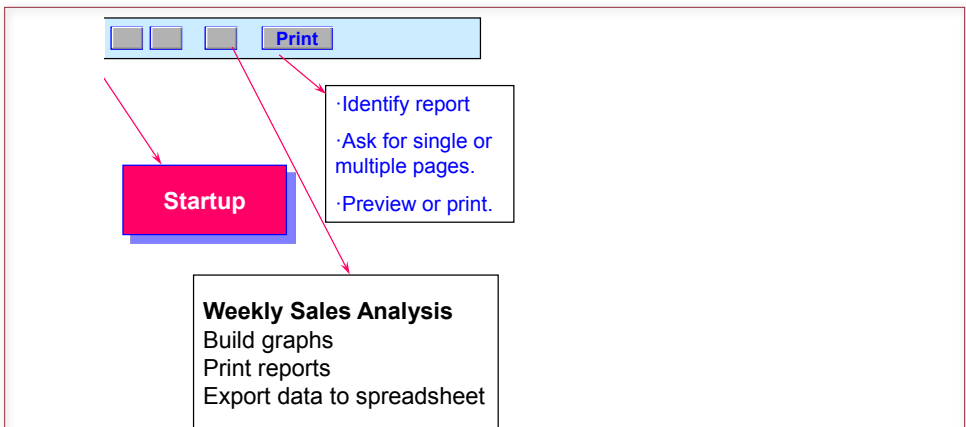


Figure 8.12

Sample toolbar. Toolbars can contain buttons and menus. Buttons generally display icons. When the pointer moves over them, a tooltip is displayed that briefly describes the button. When the button is clicked, an action is performed or a menu is displayed.

try-level users need only four or five commands, display only those options on the menu to make them easier to find. Third, you can add special functions to a custom menu. For example, you might add a special Help command to send e-mail to your support desk. Fourth, menu choices can be activated by keystrokes. Hence touch typists and visually challenged workers can use your application without looking at the screen.

Toolbars

Custom menus are usually implemented on toolbars. A toolbar contains a collection of buttons and menu items. When the user clicks a toolbar button, a predefined operation is executed. A toolbar can contain traditional buttons and textual menus. Most toolbars are **dockable**, which means that users can drag them to any place on the application window. Web-based forms rarely support dockable toolbars, but you can put menu options in a separate browser window or frame.

The purpose of a toolbar is to provide single-click access to complex actions or to commands that are used frequently. For example, many toolbars have an icon to immediately save the current work. As shown in Figure 8.12, you can put virtually any icon and any command on a toolbar. You can set different toolbars and menus for each form. You can even have multiple toolbars. For example, one toolbar might contain commands that apply to the entire application. Then special toolbars can be added as each form is opened.

Creating Menus and Toolbars

To support standardization and to simplify creating menus, most application development environments have a menu-generation feature. The exact steps depend on the system you are using; however, three basic procedures are used to create a menu: (1) Choose the layout or structure, (2) Give each option a name and an access key, and (3) Define the action to be taken when each option is selected. The main steps for creating a toolbar are similar except that you often create small graphical icons instead of text (step 2). When you create an icon, never assume

that users will recognize an icon or understand what it represents. Most systems enable you to define a tooltip for each option. When the user moves the pointer over the icon, the **tooltip**, or short comment, is displayed. Every toolbar button must have a tooltip.

Creating toolbars and menus is straightforward with recent application development systems. You can customize an existing toolbar by adding or deleting options. Similarly, you can create a new toolbar. Button icons and menus can be dragged to the toolbar. The main step is to set the properties of each item. Menu names should be short and descriptive. You should also try to follow the standard names used in commercial software. To specify the access key, precede the key letter with an ampersand. For example, the *&File* text will appear as File, and the Alt plus F keys will activate that option. Shortcut keys (e.g., Ctrl + D) can be specified in the property settings of the detail menu item or the button command.

Most systems enable you to create multiple toolbars and then activate or deactivate toolbars for different users or in different areas within the application. You generally have to create a couple lines of code to activate or deactivate a specific toolbar.

Accessibility

How can a computer application be modified for people with disabilities? Accessibility is an important question, and it is also required for any application purchased by the U.S. Federal government. With the widespread adoption of graphical user interfaces in the 1990s, many people with disabilities encountered problems using the new applications. People with good vision might see value in dragging an object from one location to another, but many operations that rely on vision are unusable by other users. Certainly users with vision challenges will have problems, but it can also be difficult for other people to control a mouse or touch screen with enough detail to select and move items on a screen.

The most common methods to improving accessibility are:

1. Support multiple input methods (keyboard as well as mouse).
2. Do not put text into graphics, and use the Alt text tag to describe all images.
3. Use default and scalable fonts, do not use fixed sizes.
4. Select user-chosen colors instead of picking your own. In Windows, use defined values such as System.ControlText.
5. If you must pick your own colors on Web sites, use a style sheet and stick with high-contrast colors.

The goal is to ensure that your forms and applications accept multiple input methods. In particular, users should be able to navigate the application by using just the keyboard. Menu and toolbar selections should include keystroke options. Typically, these selections are made by using the *Alt* key along with other mnemonic keys. Short-cuts to specific actions are usually defined by *Ctrl* key combinations. The *Tab* key should move the selection point within a form to different fields with *Back-Tab* (Shift-Tab) moving in reverse. Of course, all of these keys need to be defined—preferably in one location with a list that can be read and memorized by users.

Along the same lines, use system-defined fonts and colors within your application. Avoid hard-coding a font size (such as 11 points) or color. With existing operating systems, users can define font sizes and colors that work best for them. When you hard-code sizes and colors in your application, your choices override those of the user. They might look good to you, but they could be invisible or

highly annoying to the users. Remember that many people (as much as ten percent of the U.S. male population) can be red-green color blind. Most systems enable you to select colors based on the system-defined palette. For example, choose System.Text instead of “Black” and your application will pick up the values defined by the user.

In addition to input issues, your application needs to be conservative with graphics and images. Vision-impaired users often rely on screen readers to pick up the text from the page and read it aloud. In general, the screen readers cannot read text or interpret figures or directions written into images. When images are needed, be sure to enter text in the ALT tag that specifies in text what the image represents or critical information that it contains.

Most development systems today include tools to provide these standard features. However, typically you need to activate them and assign the keystrokes to them. As shown in Figure 8.13, menu and toolbars are often activated by adding an ampersand (&) in front of the hot key for that item. For instance an entry of *&Help* would generally be displayed with an underscore under the H as Help. The inclusion of the ampersand tells the menu system to watch for the Alt-H combination to trigger that selection. All of the necessary tools are built into the platform, but as a developer, you must enter the ampersand every time you define a menu, button, or toolbar option.

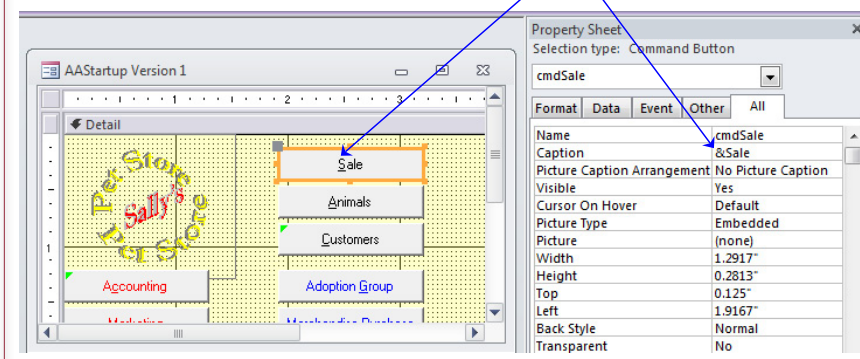
These secondary input methods are also useful for people without vision problems. Because they are triggered from the keyboard, they can improve data entry speed for almost everyone. For example, practice with Word and Excel, using keyboard combinations to select menu items and you will find that many tasks become easier and faster because you do not have to move your hands from the keyboard to move the mouse.

One of the more interesting sources of ideas for accessibility is the U.S. government. The U.S. government has been required to implement accessibility options for several years. The ruling is known as “Section508” from the number of

Figure 8.13

Specify Alt-letter combination with ampersand. Every button, menu, and toolbar option should have a keyboard definition. Many systems use the ampersand (&) method before the key letter.

Ampersand (&) Adds underline and Alt-letter trigger



the original statement. Government agencies have built an official Web site to discuss the topic: <http://www.section508.gov/>. A commercial site has similar notes: <http://www.ada508.com>.

In 2013, some discussion in the Federal government suggested that Congress might apply the same rules to commercial Web sites. The Americans with Disabilities Act (ADA) was written to require physical stores to provide access to everyone. So a few people have suggested that those with disabilities should also have equal access to any online site. It is not clear if the legislation has enough support to pass. It is also not clear that such a requirement is necessary. Presumably, some sites will find it useful to provide accessibility features. Is it truly necessary that all sites provide the same features? At the cost of opening up all Web sites to potential lawsuits. (The ADA has resulted in several nuisance lawsuits in the physical world.) But, some of the basic tenets of accessibility are straightforward and can be helpful to many users. Many of the suggestions for Web-based forms can be implemented on each page, such as including text descriptions of all functional images.

Custom Help

How do you create custom Help files? Online Help systems have grown to replace paper manuals. The goal is to provide the background information and the specific instructions that a user might need to effectively use the application system. Help files can contain text-based descriptions, figures, and hypertext links to related topics. As much as possible, the help messages should be **context sensitive**. The users should be presented with information that is designed to help with the specific task they are working on at the time. For instance, a user might want a definition of some term or more details about what actions can be performed on a specific page. Yet the Help system must also have an extensive search engine so that users can find information on any topic. Figure 8.14 illustrates a sample page from a Help system.

Microsoft has embedded a Help system within Windows for several years. This product has progressed through several versions. Most software developers use this system so users get a consistent Help system across all products. The Windows Help system displays the files, handles the links, table of contents, indexes, and searches almost automatically. As a developer, you can concentrate on creating the files that contain the basic information and the necessary links. Then the help compiler converts your data into a special file that the Windows Help system can display and search. Once you learn the basic elements of creating a page, the hard part is writing the hundreds of pages needed for a complete Help system. Most directors of large development projects hire workers just to write the Help files.

Help files built for the Windows system can be used with any application that runs on the Windows operating system. However, these compiled files do not work on the Web. If you are building an Internet-based application, you generally need to create a separate set of Help pages. The Oracle system provides its own Help compiler that you can use instead of the Windows system.

Most help systems today use HTML-based pages to display the text. Consequently, the help pages can also be used as support files for Web-based applications. However, the Windows-based systems always require some form of customization, so it is never as simple as just copying files.

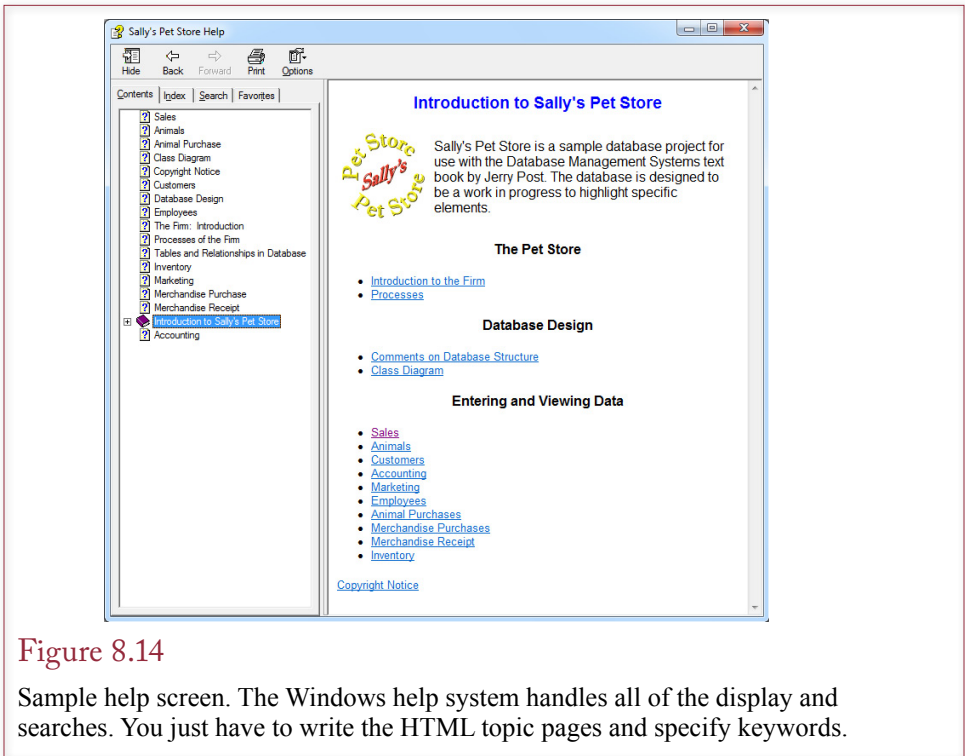


Figure 8.14

Sample help screen. The Windows help system handles all of the display and searches. You just have to write the HTML topic pages and specify keywords.

Creating a Help File for Windows

The first and most important step in creating a Help file is to understand what information a user will need. Then you must write individual pages that explain the purpose of the system and how to use it. As with any communication project, you must first understand your audience. What types of people will use the application? What is their reading level? How much experience and training do they have with computers in general? Do they understand the business operations? The goal is to provide concise help information in a format that users can quickly understand. Usability studies show that most users do not want to use the Help system—they prefer systems that are easy to use. When users turn to the Help system, they are generally in a hurry and want a simple answer to a specific question.

Once you understand the needs of the users, you can write the individual Help pages. Five basic components are used to create a Help system: (1) text messages, (2) images, (3) hypertext links between topics, (4) keywords that describe each page, and (5) a topic name and a number for each page.

Microsoft currently supports two Help systems and is developing a third. But the company has experimented with several versions and it is not clear if newer versions are going to be carried forward. The original system generated HLP files. The second and most common one generates CHM files. A new system was designed for use with Visual Studio 2010. Its files are ZIP archives with a suffix of mshc. It is an improvement over Microsoft Help 2, which was used for Visual Studio 2003/2005 and Office 2007 (and 2010); but has been discontinued. The newer Microsoft Help Viewer (mshc) version has some useful features, but it might be a while before it is more widely implemented.

```

<object type="application/x-oleobject"
classid="clsid:1e2a7bd0-dab9-11d0-b93a-00c04fc99f9e">
  <param name="Keyword" value="Contents">
  < param name="Keyword" value="Introduction">
  < param name="Keyword" value="Sally's Pet Store">
  < param name="Keyword" value="Management">
</object>
<html><head>
<title>Sally's Pet Store Introduction</title>
<link rel="stylesheet" type="text/css" href="PetHelpStyle.css" />
</head><body>
<h1>Introduction to Sally's Pet Store</h1>
<table><tr>
<td><img src='PetStoreLogo2.gif' border='0'></td>
<td>Sally's Pet Store is a sample database project for use with the
Database Management Systems textbook by Jerry Post. The database
is designed to be a work in progress to highlight specific elements.</td>
</tr></table>
<h2>The Pet Store</h2>
<ul>
<li><a href='FirmIntroduction.html'>Introduction to the Firm</a></li>
<li><a href='FirmProcesses.html'>Processes</a></li>
</ul>
</body></html>

```

Figure 8.15

Partial sample Help page. Create each topic as a separate Web page using HTML. The anchor `<a>` tag links to other pages. The `` tag loads images. Use style sheets to set fonts and design. Use a table or a style to control layout. Place keywords for the page in the `<object>` tag.

The discussion in this section focuses on the CHM approach because it uses **hypertext markup language (HTML)** files—which are relatively easy to convert to Web based help. The newer Help Viewer system also relies on HTML files (technically well-formed XHTML pages), so the base concept is the same. Writing help files in HTML is relatively easy and many good tools exist for creating Web pages. However, be careful with the tools: Some of them, such as Microsoft Word, create complex code that might not work well with the Help compiler. You want to use an HTML editor that produces basic HTML code without relying on XML or JavaScript.

From a design perspective, it is crucial that you first design a style for your Help system and define that style using a cascading style sheet. A **style sheet** sets the typeface, font size, colors, and margins. The power of a style sheet is that you define all of the layout options in one place. Each page linked to the style sheet picks up those styles. So when you want to change the entire layout of your Help file, you make a few changes to the style sheet and every page uses that style.

Every topic is created as a separate HTML page. Users will be shown one page of material at a time. Try to keep topics short so they fit on one screen. Each Help page will contain links to other topics. Figure 8.15 shows part of a basic Help topic. Each page should have a title (marked with the `<title>` tag). Pages generally have links to other topics (using the HTML standard `<a href>` tag). Images can

be in one of two formats: joint photographic experts group (JPEG) and graphics interchange file (GIF). Most Help images will be line-art drawings and should be in the GIF format. Most graphics packages can create and store files in these formats. When you save the file, use only letters and numbers in the filename—do not include spaces. Because you will eventually have hundreds of pages, it is a good idea to keep a separate list of the pages along with a short description of the topic and when it was last modified.

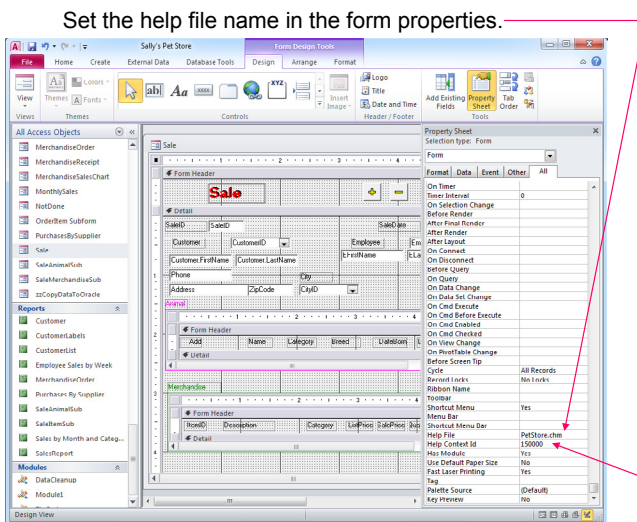
Keywords are an important part of every Help page. They are used to create an index for the user. An index lists the keywords alphabetically, when a user double-clicks a word, the corresponding Help page is displayed. The best way to create keywords is to enter them on each topic page. The easiest method is to copy the code from Figure 8.15 with the <object> tags and then change the keywords within that list for each page. Each keyword is listed with a separate <param> tag. If you want multiple levels, you can use a comma to list the hierarchy. For example, the three entries: (1) Sales; (2) Sales, Merchandise; and (3) Sales, Animal will create an index entry of Sales, followed by two indented lines for Animal and Merchandise.

Context-Sensitive Help

Consider an example of using help. Users working on the Sales form in your application do not want to wade through several Help pages or try to think of search terms. Instead, when they press the Help key, they expect to see information on that particular form. At a minimum, you need to create different Help pages for each form in your application. But now you need some method in your database

Figure 8.16

Setting context-sensitive help. In every form, enter the name of the Help file in the Help File property. Then enter the topic number for that form in the Help Context ID property. Every control or subform can also have a different Help topic—just enter the corresponding topic number.



```
#define PetStoreIntro          100
#define Accounting            10000
#define Animal                20000
#define AnimalDonation        30000
#define ClassDiagram          40000
#define Copyright             50000
#define Customer              60000
#define DatabaseDesign        70000
#define Employee              80000
#define FirmIntroduction       90000
#define FirmProcesses         100000
#define Inventory             110000
#define Marketing             120000
#define MerchandisePurchases  130000
#define MerchandiseReceipt    140000
#define Sale                  150000
```

Figure 8.17

Map file. Applications refer to topics by number, but the help system uses the filename. The map file (Topics.h) is a simple text file that assigns a number to each page.

application to specify which Help page should be displayed for each form. As shown in Figure 8.16, each form has properties for Help File and Help Context ID. Oracle and Visual Basic forms have similar properties. You enter the name of the file (e.g., PetStore.chm) in the Help File property. The Help Context ID requires a number. This number is a long integer and can range from 1 to more than 2 billion.

It is crucial to note that applications require a topic number, but your Help file refers to pages by their filename—not by numbers. To get these two systems to match, you must assign a unique number to every topic page. With HTML Help, you create a separate text file (usually called Topics.h) that maps this relationship. A sample file is shown in Figure 8.17. You can choose any number, but it is easier to remember them if you assign the numbers in groups. Also, with 2 billion numbers available, you can leave large gaps between the group numbers. For example, it is better to number by hundred thousands or millions instead of by ones (1, 2, 3, and so on). A useful technique is to assign numbers by business object (e.g., all Customer Help files might be numbered from 1,000,000 to 2,000,000). Once you have created the file, use the HtmlHelp API Information button (left side, fourth from the top) to tell the Help Workshop to include the file. Now, go through every form in your application and specify the file name and topic number for that form. Avoid changing the topic numbers in the Help file; they are hard to find in your application.

After you have created all of the files, you need to run the HTML Help compiler to combine everything into a single CHM file. A version of this tool can be downloaded free from Microsoft. Search for the `htmlhelp.exe` file. However, it is a relatively limited tool that can be cumbersome to use. Most development teams purchase a commercial product to gain more features including support for multiple writers. Several commercial tools exist at varying prices, and they generally include features such as support for multiple file types and version control.

```

<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>My Page Title</title>
  <meta name="Microsoft.Help.TopicLocale" content="en-us" />
  <meta name="Microsoft.Help.TopicVersion" content="100" />
  <meta name="Microsoft.Help.Id" content="fadf1f04-77dd-43fb-81f6-72e5ae0bfc3d" />
  <meta name="SelfBranded" content="true" />
  <meta name="Microsoft.Help.Locale" content="en-us" />
  <meta name="Microsoft.Help.Package" content="My_Help_Package_Pets_en-us_1" />
  <meta name="Microsoft.Help.F1" content="PetStore" />
  <meta name="Microsoft.TocParent" content="-1" />
  <meta name="Microsoft.Help.Category" content="Petstore::Introduction" />
  <meta name="Microsoft.Help.ContentType" content="Concepts" />
  <meta name="Microsoft.Help.Keywords" content="Introduction" />
  <meta name="Microsoft.Help.Keywords" content="Pet Store" />
  <meta name="Description" content="Basic description goes here..." />
  <meta name="Microsoft.Help.tocOrder" content="1" />
</head>
<body class="primary-mtps-offline-document">
  <div class="topic">
    <div class="majorTitle">This is the Page Title</div>
    <p>Sally's Pet Store .... </p>
    <p><a href="ms-xhelp:///?Id= fadf1f04-77dd-43fb-81f6-433e3ae08ac22">My Link</a>

  </div>
</body>
</html>

```

Figure 8.18

Sample HTML Help 3. Meta tags within the file are used to define the basic features such as title, ID, table of contents location, and key words. Links use an ms-xhelp format to specify the ID of the link page.

Windows Help 3/Help Viewer

Microsoft might be changing the Windows Help system. Keep in mind that the company has tried at least at least two other times to create a new help system. The current version, loosely known as Help Viewer or Help 3 is an improvement over Help 2. Currently, the tool is only used to create help files that work within the Microsoft Visual Studio tool. However, there is a chance that the tool could be applied to other products in the future.

In terms of writing Help files for Help Viewer, the process is similar to the existing HTML Help: Begin by writing each topic in a separate HTML file. One important catch is that the HTML file actually needs to be XHTML—which is a more precise version of HTML that is compatible with XML. The headers are slightly different, and all tags must be complete. For example, a paragraph must have both a beginning and ending tag: `<p>My paragraph</p>`.

The other big difference is that all metadata is stored in the same file. There are no separate files for topics, keywords, table of content lists, or keywords. Everything is marked in the page using special tags. Figure 8.18 shows the basic format of a simple XHTML help page. Note the use of meta tags to specify the items

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>An optional title.</title>
</head>
<body class="vendor-book">
  <div class="details">
    <span class="vendor">Pet Store</span>
    <span class="locale">en-us</span>
    <span class="product">Pet Store Sales </span>
    <span class="name">Pet Store</span>
  </div>
  <div class="package-list">
    <div class="package">
      <span class="name">package1</span>
      <a class="current-link"
        href="packages\package1.mshc">package1.mshc</a>
    </div>
    <div class="package">
      <span class="name">package2</span>
      <a class="current-link"
        href="packages\package2.mshc">package2.mshc</a>
    </div>
  </div>
</body>
</html>
```

Figure 8.19

Sample manifest file. A package is a single mshc help file. Name the entire manifest to: helpcontentsetup.msha and place it into the help archive folder.

needed to create the help file—notably the TOC specification and the key words. Entering a TOC value of -1 indicates that the entry (title) will be placed at the top of the hierarchy. To place an item lower in the hierarchy, simply enter the Help.ID value specified in the parent.

The nice thing about the new format is that you no longer need a help compiler to create the final help file. Simply create a new ZIP archive and place all of the text and image files in that compressed folder. Add a manifest file (helpcontentsetup.msha) and rename the archive from .ZIP to .MSHC. Figure 8.19 shows a sample manifest file with links for two “packages” or mshc files. Be sure to specify the names and locations correctly. For instance, the sample file refers to the pages stored within a “packages” subfolder.

At this point in time, your file will probably not open because Windows (and Office) are not set up for the new format. You might be able to use the HelpLib-Manager.exe program to install your new file and test it. Eventually, either the new help system will be adopted and integrated into Windows, or discarded for something newer (again). Either way, the hard part of creating Help files is identifying the topics and writing text that will actually benefit the users. These HTML files can be used for either of the current versions of HTML Help as well as stand-alone help files on Web sites.

Handling Errors

What does your application do when something goes wrong? Error handling is a task that is often relegated to coding on individual forms. However, it is a critical step—particularly in terms of security—so you need to review it at this stage of development. Also, error handling should be consistent across the application to avoid confusing users. At the same time, you need to create a logging facility so that runtime errors in the application are recorded and reviewed periodically so the application can be improved.

In terms of security, it is critical that your application catch all errors. Without special handling error messages can crash the entire application. Worse, they can lead to overwritten code providing an opportunity for criminals to take control of a machine. Even relying on the system error handling is dangerous, because default error messages often provide information that can be used to attack your application.

Catching Errors

Most development languages provide commands to trap runtime errors. Most of them use a variation of the try/catch syntax. The code to be protected is run within a try section. If an error occurs, execution is transferred to the catch section. Your error-handling code can look at different types of errors and handle them separately, or simply treat all errors the same. Your ultimate mission is to devise error handling code that can automatically deal with common problems. The intelligence built into error-handling code is one way to tell the difference between amateurs and professional developers. Usually, you need users to help create errors so you know what to expect and the best way to handle them. The need for user testing is one of the reasons complex error handling is added at this stage of the application development.

Figure 8.20 shows the basic syntax for several programming systems. Most use a try/catch approach but the syntax varies. Note that Visual Basic is the same as C# but Basic does not use the braces. The basic structure includes a section

Figure 8.20

Common error-handling structure. Most systems use a try/catch structure but use different syntax to define the sections. The SQL 2003 standard supports various conditions (SQLEXCEPTION) and can EXIT the existing code or return to it (CONTINUE) after processing the handler code.

Oracle	SQL Server	C#	Access
BEGIN {code} EXCEPTION WHEN OTHERS THEN {code} END	BEGIN TRY {code} END TRY BEGIN CATCH {code} END CATCH	try { {code} } catch (exception e) { {code} }	ON ERROR GOTO errX {code} exitX: Exit Sub errX: {code} End Sub
SQL 2003 Standard	DECLARE EXIT HANDLER FOR SQLEXCEPTION Sql_procedure_name {code}		

or routine that is executed when an error arises. You have to make sure that each procedural code section is covered by at least one statement that directs errors to a handler. Then you can write code to identify the specific error and find ways to solve the problem or send a message to the user and exit gracefully.

One of the challenges of database programming is that procedural code can exist in two places: (1) Within procedures on the server, and (2) In routines such as C# that run on the client computer. You need to examine the application to be sure that both types of code are protected by error-handling routines. Errors that are trapped within database procedure code need to be returned to the client system to perform additional error handling, including displaying warnings to the user.

Logging Errors

An important step in trapping errors is to record them. Yes, it is helpful to display problems to the users; but users generally cannot do anything to solve the problem. You need to create a routine that inserts the error message, location, and date into a special table. Each error-handling routine should include a line of code to call this logging procedure. You might want to include additional data, such as values of key local variables, for complex procedures.

When the system has been running in production for a while, you can retrieve the values from the error-logging table. A simple query will show you which code sections cause the most problems and help identify the types of mistakes encountered by users. You use this information to fix code errors and write more intelligent error-handling code. The ultimate goal is to prevent users from having to deal with run-time errors. Your code should be able to identify and handle the main issues automatically. Of course, you cannot solve every problem—such as hardware or network failures—but you can identify them and give advice to the user.

Debugging

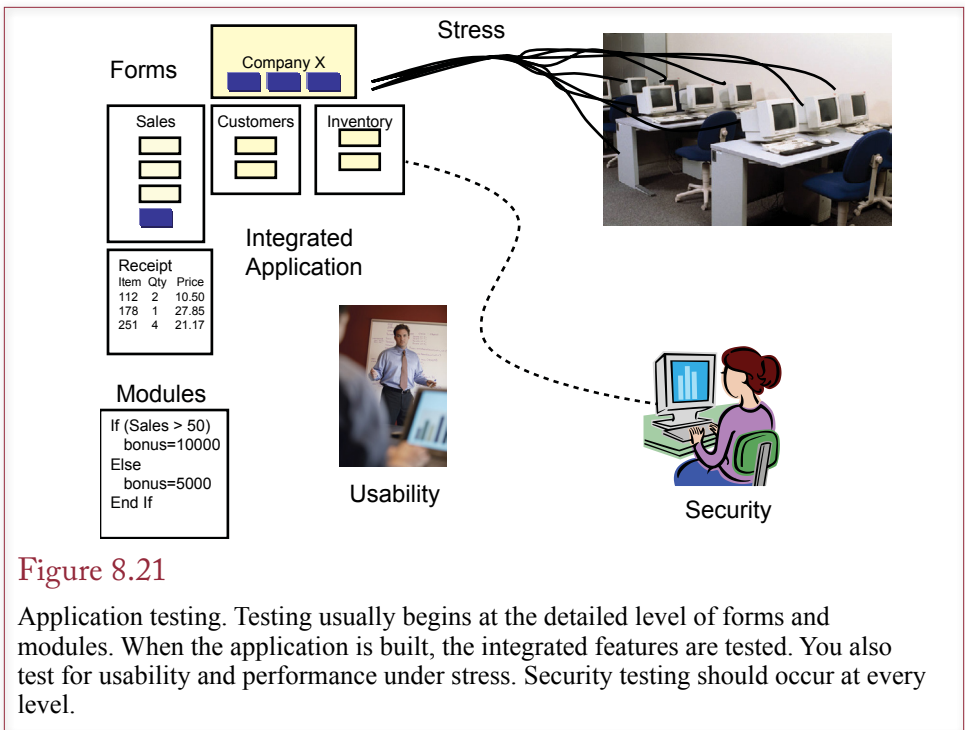
Once you know the approximate location of an error, you need to track down the cause. Fortunately, most contemporary systems have interactive debugging tools that enable you to set break points and step through the application code line by line. You can examine the values of local variables and even test queries.

The debugging process is more complicated when you have code running on a server, and considerably more difficult when code runs on multiple tiers including servers, clients, and middle-tier systems. Adding multiple levels requires you to track down the true location of an error. Depending on your tools and the final configuration, it is more difficult to run debuggers on multiple levels.

In many cases, you will have to resort to older debugging methods, such as adding debugging print lines to your procedures that report the current values of key local variables. In multi-tier systems, pay particular attention to the timing of events including the code and when variables are initialized and assigned values.

Testing

How do you know your application works correctly? Every application needs to be tested before it is turned over to users in a production environment. As indicated in Figure 8.21, many levels of tests can be performed, but ultimately, you can never catch all of the errors. Your goal is to find as many of the errors as possible with the time and money available. Keep in mind that errors caught earlier are easier and less expensive to fix, and it is better to catch errors before they cause expensive problems for users. Larger systems with multiple developers will



require special groups of testers dedicated to finding problems. In smaller systems, you might have to test your own work. In either case, you should enlist the assistance of actual users who will always try things that never occurred to you. In a twist, test-based development is a modern approach to development where test cases are created first. Then whenever code is written or changed, the test cases are automatically rerun to ensure the code still works correctly. Several tools have been developed to help automate the testing process, but it still requires considerable time to develop all of the test cases.

Form and Module Testing

The most basic level of testing occurs when you create modules and forms. Any query, report, procedure, or section of code that you create should be tested as it is written. When you first create an object, you should understand its primary purpose and have sample test data to ensure that it works correctly. In particular, if you need to perform complex calculations or logic, you need to work with users to develop suitable test cases. These test cases should be stored and reevaluated at each testing point. Some organizations use pairs of developers, where one person is responsible for collecting test cases and continually testing sections of the project as it is being built.

You should also integrate the testing with form validation. Forms should restrict the data that users can enter to reduce the possibility of bad data, or even intentional attacks on your application. Where possible, you should use drop down lists and option choices so users do not have to type in values. When users do enter values by hand, you should include validation rules on the form to provide immediate feedback to the users so the data can be correct as close to the source as possible.

Integrated Application Testing

Once the overall structure of the application has been created, it needs to be tested as a complete unit. In particular, you need to ensure that data is passed correctly across forms, modules, and reports. Any forms that contain links need to be tested. For instance, linking a Sales Order form to the Customer form should result in displaying the details for the customer currently selected on the Sales form. But you also need to test extreme conditions. What happens if no customer has been selected yet and a user clicks the link button? Likewise, what happens if the user tries to print a blank receipt report? Be sure that the application continues to run even if absurd choices are made. Verify that data is being stored properly, and that security conditions are being maintained.

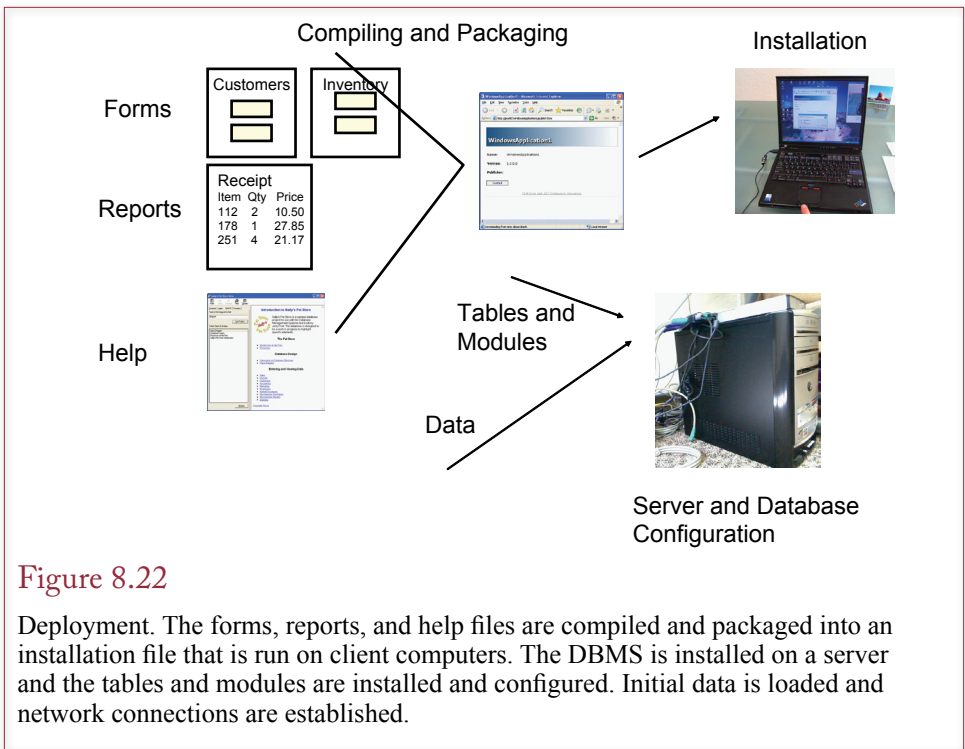
Stress or Performance Testing

Many developers and companies have encountered problems when an application hits the real world. Forms and reports that run fine on the developer's server die a slow death when pushed out to thousands of users with millions of rows of data. Unfortunately, systems performance is not always linear. For example, a task with 10 users might require 2 seconds to run; but you cannot claim that moving to 100 users will result in 20 seconds. More likely, instead of increasing by ten times, the time will increase exponentially, requiring 40, 50, or even 60 seconds. Some systems are more scalable than others—meaning that performance can be improved by adding hardware capacity and the process is close to linear. Other systems are more complex, but either way, you need to stress test the application to find out what will happen.

The challenge is that it is difficult to test big applications with thousands of users—without actually implementing the system. Where are you going to find the hardware and the thousands of users to test the system with sample data? Some companies sell tools that help stress test an application. The tools automatically generate transactions and send them through your application. You can increase the load on the servers by using only a few automated client computers. You can also test the servers and networks on a smaller scale by throttling down the hardware and networks. Instead of pushing 1,000 transactions through a 100 mbps network, you could test with 100 transactions across a 10 mbps network. It will not give exact results, but it will help you see what happens if a key connection gets overloaded. This test is particularly useful for connections from the Web server to the database server.

Usability Testing

In addition to testing for accuracy, errors, and performance, you also need to ensure that the system performs the tasks that users need. As part of the process, you need to have actual users work with the system. You need to be sure that users understand the forms and the process. The system needs to be easy enough to use so that it does not require huge amounts of training. It also needs to be efficient so that users do not waste time entering unnecessary data or searching for information. A developer can spend hundreds of hours building forms and applications. At some point, everything seems easy. You need the fresh perspective of actual users to identify bottlenecks and other issues. At one level, development is much like artistic design. Developers make dozens of choices when building applications. How do you make the best choice? The answer is that usability needs to become a key component right from the start. And the application specifically needs to be tested for usability.



Usability testing also needs to include testing the accessibility features. First, someone needs to go through the entire application and ensure all of the features are activated. It is too easy for a developer to forget to tag a button, menu, or toolbar so it is accessible with a keyboard. So someone needs to go through every item on every page and verify that accessibility is activated. Also, whenever possible, it would be useful to have someone with accessibility issues to actually use the application. A real-world test can provide valuable insight into the application flow, terminology issues, or other potential problems.

Security Testing

Security concepts are explored in other chapters, but companies have learned that security also needs to be addressed throughout the development process. It is not something that is added on at the end of the design. Testing for security issues includes some of the basic tests—particularly validation and module testing. It includes checking user input for common SQL injection attacks. A **SQL injection** attack consists of an attacker entering malicious SQL code into a text box in your application that replaces your intended SQL statement. The classic example is creating your own login screen and allowing users to enter any text as a username and password. The problem is compounded when you use string concatenation to build the SQL query. You should always use parameterized queries instead of string concatenation. More importantly, you should never trust anything entered by a user—and always restrict or validate what they are allowed to enter.

Consider the simple login example, where your application retrieves a `UserNameText` and `PasswordText` variable from the input screen. It is tempting to write the simple lookup query: “SELECT UserID FROM UserList WHERE User-

name=' ' + UsernameText + " ' AND Password=' ' + PasswordText + " ' ". Ignoring the fact that the password should be encrypted, this query will work fine as long as users enter legitimate values. However, what happens when an attacker enters a special SQL string for the UsernameText: ' OR 1=1 --. Plug this value in and write out the SELECT statement. The quotation mark closes the first one, the OR statement is always true, and the two dashes comment out the rest of the SQL command. As a result, the query will always return valid UserID and the attacker will be logged into your system. Worse, it is possible to write more complex SQL statements that do nastier things, such as retrieving all of the data from the UserList table, or even deleting tables in your database. However, all of the SQL injection attacks have a common element. They include the single quotation mark to close the required opening quote, and they use the double hyphen comment mark. The simplest solution is to test all input code and remove or change quotation marks and double hyphens to spaces. Whenever possible, you should restrict the length of data entered by users to prevent someone from writing long, dangerous code. Of course, restricting inputs can impact normal data entry, such as handling the name O'Brian which contains an apostrophe or single quote character.

Security testing also involves testing the entire application—including steps that might not be computerized. For example, how are passwords generated? What happens if a user loses a password—how is it reset? Is this process secure and logged? When the integrated application is being tested, you should also include basic security tests—particularly bad data that includes excessively long values and SQL injection elements. For large projects, at least one person should be assigned to attack the application, listing potential threats and methods that might be used to obtain unauthorized access.

Deploying an Application

How will your application be installed? As shown in Figure 8.22, once you have developed an application, you must collect all of the associated files (e.g., database, system, forms, reports, and help) and distribute them to users or install them on servers. You must also implement security precautions and assign user access rights. The details depend on the type of application system, whether the users are employees, and the size of the application. It is usually easiest to install applications on a server in one location. Even if you need a separate application (Web) server, installation and maintenance are relatively easy when the files and databases are in one location. If your application needs to install elements on client computers, several additional steps are needed.

Packaging Files

One of the first steps is to identify and collect all of the files. These primarily consist of the forms, reports, and help files. With a small application, built by one or two developers, it is relatively easy to identify and collect all of the files. With large applications that include hundreds of forms and reports, you need a version control system to name each file and track the versions and changes.

Some systems store forms and reports internally, some treat them as separate files, and a few compile them into a set of executable files. The method of packaging the files varies in each case, but it must still be done. You also need to test the resulting system to ensure all of the files are included and have the correct names. As much as possible, you need to automate the build process. Some systems include an automatic build, in other cases you will have to write script files. Either

way, it is important to automate the steps because you will have to rebuild the application many times, and it is too easy to forget something. Scripts are easy to modify to avoid mistakes.

Installation Programs

If you are going to put any portion of the application on client computers, you need to use an installation program to automate the installation. Several tools exist, some versions are included with the DBMS and other versions are sold by independent vendors. Installation programs bundle the various files, check the target system for prerequisite files, and handle all configuration changes. Most of the installation tools support packages delivered on CD or downloads from a Web server. Some of the newer tools, including the one with Visual Studio, can be installed directly from a Web site and check for updates. The installation system also has to configure the database connections so the client component can attach to the database server.

Microsoft Access adds more complications to the installation process. In its most common form, the client computers will each need a licensed copy of the Microsoft Access software. You will also want to encrypt the database forms and reports to prevent users from changing them. In most cases, you will want to split the database into two pieces. Details are provided in the Access Workbook. Microsoft provides another alternative if you do not want to install the full copy of Access on each client computer. You can purchase the Access Developer kit which includes a runtime module. The installation system can install the runtime module so that your application will run without requiring a full copy of Microsoft Access.

Server and Database Configuration

An application also needs the servers and databases configured. The best way to handle the database configuration and base data loading is to write SQL scripts. You can create the scripts as the application is developed and tested. The scripts make it easy to load a new copy onto a test server. More importantly, they can be used to create a backup server or to reinstall the application if something goes seriously wrong. The applications associated with these books use script files, and you can use them as a template for your own applications. Even if you believe an application will only be installed one time, you should create the server script files. You will be surprised at how many times you will need to delete and reinstall an application while it is being tested.

Summary

An application is a collection of forms and reports designed to function as a system for a specific user task. Applications must be easy to use and designed to match the tasks of the users. Application design begins with the overall structure, which is often held together with startup forms. Menus and toolbars add structure to the application by providing commands that are common to the entire application. Toolbars can also be created for specific tasks and individual forms. A context-sensitive Help system with both general descriptions and detailed help notes is crucial to creating a useful application. Most applications also need to define individual transactions so that related changes will succeed or fail together.

You need to add error handling to all forms and modules and perform several levels of tests, including performance and security testing. You need to create a relatively automated approach to deploying the application—particularly if it

needs to be installed on client computers. Several installation tools exist to package the files and support automated installation. For the database and server-based code, forms, and reports, you need to create SQL scripts that will create tables and load the basic data.

A Developer's View



Miranda is learning that applications are useful only if they make the user's job easier. A good application is more than just a collection of tables and forms. That means you have to organize the application by the tasks of the user. You also need to add help files and toolbars. You need to add error-handling code to your application. Once the application is fully tested, you need to create an installation package. For your class project, you should create the overall application structure (switchboard forms, interlocking forms, toolbars, help files, and so on). You should build and test the scripts and installation setup.

Key Terms


accessibility
administrative tasks
application
back end
cascading style sheet
context sensitive help
dockable
error handling
front end
help system

hypertext markup language (HTML)
menu
middle tier
prototype
SQL injection attack
style sheet
startup form
template
toolbars
tooltip

Review Questions

1. What are the fundamental principles to follow when designing an application's structure?
2. How does the purpose of an application (transaction processing, decision support, or expert system) affect the design?
-  3. How are startup forms commonly used?
4. What are the potential problems with startup forms?
5. What is the purpose of menus and toolbars in an application?
6. What features are needed to make an application more accessible?
7. What are the primary steps involved in creating a context-sensitive help file?
8. What are the major methods for handling runtime errors in an application?
-  9. What are the primary forms of testing?
10. What are the main steps in deploying an application?


Exercises

1. Find examples of two input forms—such as Web applications or business forms. Compare the applications on design and functionality. Explain the similarities and differences.
2. Find a Web site that has a separate mobile-based application. Explain the similarities and differences between the two types of forms. What features had to be sacrificed to make the mobile form? What choices would you have made differently?
3. HTML5 supports graphical actions, although the built-in capabilities are somewhat primitive. Assuming you have programmers to create them, design a new Web-based process to purchase items and handle shopping carts that use graphics and drag-and-drop elements. Just sketch the concepts—it is not necessary to create them.
4. Create a custom toolbar menu with at least two icons and two drop-down menus that include at least three options each.
-  5. Briefly explain how a touch-based menu would be different from a mouse-based menu.
6. Application menus can have many options. Briefly explain how you would solve the question of identifying the structure and items on menus.
7. Create a small custom help file that contains three pages of help. Create a form and assign the help key to open one of the help topics.
8. Examine at least three Web sites and explore their help sections. Briefly compare similarities and differences among the three sites. Explain which features you would use in your own applications.

9. Write a function to log runtime errors to a special database table or a file. Create a form with a button that contains error-handling code. When the button is pressed, it should trigger a runtime error (e.g., divide-by-zero), and call the logging function to save the error message.
10. Research and briefly describe the test-based development methodology and explain how it could be used in database applications.




Sally's Pet Store

11. Find at least two Web sites for pet stores and compare them. Select the primary features that you would want to use for a site for the Pet Store. Briefly explain how you would improve and differentiate your site.
 12. Design and create a menu system and toolbars for the Pet Store database that would be used by clerks and managers in the store.
 13. Design a template for the input forms. At a minimum, specify colors, fonts, and page layout. Rebuild at least two of the forms in the new template to test the styles.
 14. Create and write initial help files for the Pet Store. Include at least three new pages of help, the table of contents, and keywords.
 15. Find a user (non-CS and non-IS) who can test the application. Observe the user's progress and identify any problems or issues that arise. Describe changes you would make to improve the application.
-  16. Assuming the store is going to use the finished application, outline a plan to install and deploy it in the store on a single computer.



Rolling Thunder Bicycles

17. Examine the Rolling Thunder Bicycles application and outline the menu structure by checking the forms and reading the help file.
-  18. Explain how the list box is used to handle receipt of merchandise from suppliers. Outline the process that is used to tie the receipt to the purchase order.
19. Outline a plan for stress testing the application. Begin by identifying where the application will be used and how many people will likely use it at one time.
 20. Design a new toolbar or menu that supports operations by categories of users: Managers, Order-clerks, Production, and Finance/Accounting. You can just sketch the toolbars/ribbons instead of actually building them.
 21. Work through the application and test it for accessibility. Identify any changes that need to be made.


Corner Med

22. Design a menu or toolbar for Corner Med to make it easy to use within the clinics.
23. Identify potential application problems and failures that might arise and outline a plan to handle them. (Focus on software, not hardware or networks.)
24. Write the deployment plan for the application, assuming there will be one workstation at the central check-in desk and one in each physician office. Typically, there are three to five offices per location.
25. Design the Patient Visit form so it can be used on a mobile tablet with a 10-inch screen.

Web Site References

http://www.microsoft.com/enable/	Microsoft site for accessibility issues.
http://msdn.microsoft.com/windowsvista/uxguide	Microsoft design guide for Windows Vista.
http://www.sigapp.org/	Association for Computing Machinery: Special Interest Group on Applied Computing.
http://oraclea2z.blogspot.com/	Oracle application tips.
http://www.useit.com	Web site run by Jakob Nielsen (a researcher in usability).
http://www.helpwaregroup.com/	Help authoring utilities
http://www.section508.gov	Federal government accessibility guidelines and blog.
http://www.w3.org/WAI	W3C (Web governance group) on the Web Accessibility Initiative.

Additional Reading

- Cooper, A. *About Face: The Essentials of User Interface Design*. Foster City, CA: IDG Books, 1997. [A good discussion of various design issues.]
- Ivory, M. and M. Hearst, The State of the Art in Automating Usability, *Communications of the ACM*, 33(4), December 2001, 470-516. [General discussion on evaluating system usability.]
- Raskin, J. *Humane Interface, The: New Directions for Designing Interactive Systems*, Reading, MA: Addison-Wesley, 2000. [The need for a new interface as explained by the creator of the Apple Macintosh project.]