CHAPTER 2

Systems Analysis and Database Requirements

Abstract

This chapter discusses the role of database design and development as part of a complete systems analysis and design project. Topics include the structured systems analysis and design cycle along with alternative system developments methods (prototyping, spiral methodology, and object-oriented analysis and design).

Keywords

databases
systems analysis and design
change management
structured systems analysis and design
prototyping
spiral methodology
object-oriented analysis and design

As you will discover, a large measure of what constitutes the "right" or "correct" database design for an organization depends on the needs of that organization. Unless you understand the meaning and uses of data in the database environment, you can't produce even an adequate database design.

The process of discovering the meaning of a database environment and the needs of the users of that data is known as *systems analysis*. It is part of a larger process that involves the creation of an information system from the initial conception all the way through the evaluation of the newly implemented system. Although a database designer may indeed be involved in the design of the overall system, for the most part, the database designer is concerned with understanding how the database environment works and, therefore, in the results of the systems analysis.

Many database design books pay little heed to how the database requirements come to be. In some cases, they seem to appear out of thin air! Clearly, that is not the case. A systems analysis is an essential precursor to the database design process, and it benefits a database designer to be familiar with how an analysis is conducted and what it produces. In this chapter, you will be introduced to a classic process for conducting a systems analysis. The final section of the chapter provides an overview two alternative methods.

The intent of this chapter is not to turn you into a systems analyst—it would take a college course and some years of on-the-job experience to do that—but to give you a feeling of what happens before the database designer goes to work.

Dealing with Resistance to Change

Before we look at systems analysis methodologies, there is one additional very important thing we need to consider: A new system or modifications to an existing system may represent significant change in the work environment, something we humans usually don't accept easily. We become comfortable with the way we operate and any change creates some level of discomfort. (How much discomfort depends, of course, on the individual.)

Even the best designed and implemented information system will be useless if users don't accept it. This means that as well as discovering what the data management needs of the organization happen to be, those in charge of system change need to be very sensitive to how users react to modifications.

The simplest way to handle resistance to change is to understand that if people have a stake in the change, then they will be personally invested in seeing that the change succeeds. Many of the needs assessment techniques that you read about in this chapter can foster that type of involvement: Ask the users what they need and really listen to what they are saying. Show the users how you are implementing what they need. For those requests that you can't satisfy, explain why they are infeasible. Users who feel that they matter, that their input is valued, are far more likely to support procedural changes that may accompany a new information system.

There are also a number of theoretical models for managing change, including the following

- ADKAR: ADKAR's five components make up the model's name's acronym:
  - **A**wareness: Make users aware of why there must be a change.
  - **D**esire: Involve and educate users so that they have a desire to be part of the change process.
  - **K**nowledge: Educate users and system development personnel in the process of making the change.
  - **A**bility: Ensure that users and system development personnel have the skills necessary to implement the change. This may include training IT staff in using new development tools and training users to use the new system.
  - **R**einforcement: Continue follow-up after the system is implemented to ensure that the new system continues to be used as intended.
- Unfreeze–Change–Refreeze: This is a three stage model with the following components:
  - Unfreezing: Overcome inertia by getting those who will be affected by the change to understand the need for change and to take steps to accept it.
  - Change: Implement the change, recognizing that users may be uneasy with new software and procedures.

❑ Change: Implement the change, recognizing that users may be uneasy with new software and procedures.
❑ Refreeze: Take actions to ensure that users are as comfortable with the new system as they were with the one it replaced.

The Structured Design Life Cycle

The classic method for developing an information system is known as the *structured design life cycle*. It works best in environments where it is possible specify the requirements of a system, where the requirements are fairly well known, before developing the system.

There are several ways to describe the steps in the structured design life cycle. Typically, the process includes the following activities:

1. Conduct a needs assessment to determine what the new or modified system should do. (This is the portion of the process typically known as a *systems analysis*.)
2. Assess the feasibility of implementing the new/modified system.
3. Generate a set of alternative plans for the new/modified system. (At this point, the team involved with designing and developing the system usually prepares a *requirements document*, which contains specifications of the requirements of the system, the feasibility analysis, and system development alternatives.)
4. Evaluate the alternatives and choose one for implementation.
5. Design the system.
6. Develop and test the system.
7. Implement the system. This includes various strategies for phasing out any existing system.
8. Evaluate the system.

The reason the preceding process is known as a "cycle" is that when you finish Step 8, you go right back to Step 1 to modify the system to handle any problems identified in the evaluation (see Figure 2.1). If no problems were found during the evaluation, then you wait a while and evaluate again. However, the process is also sometimes called the *waterfall method* because the project falls from one step to another, like the waterfall in Figure 2.2.

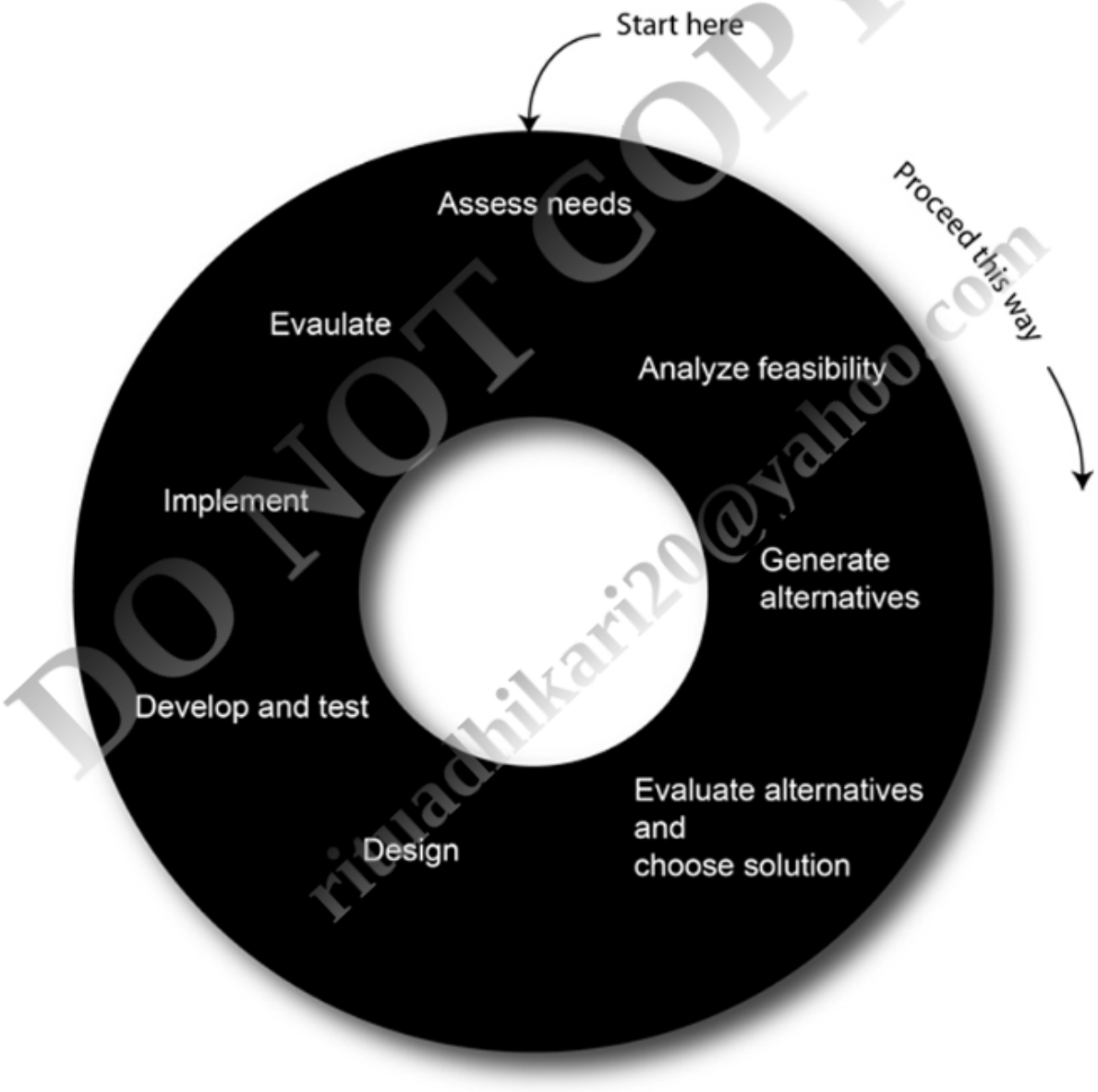

FIGURE 2.1 The traditional systems development life cycle.

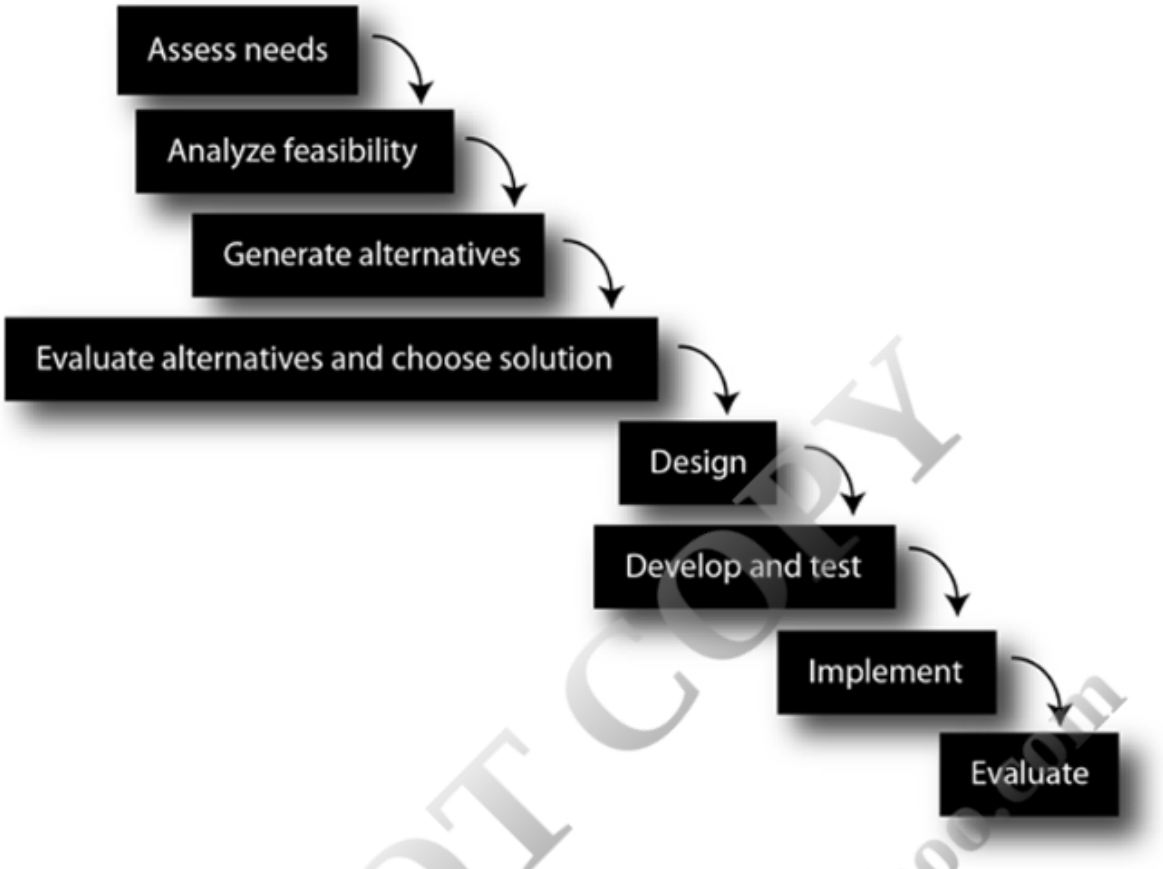

FIGURE 2.2 The waterfall view of the traditional systems development life cycle.

Database designers are typically involved in Steps 1 through 5. The database design itself takes place during Step 5, but that process can't occur until the preceding steps are completed.

Conducting the Needs Assessment

The needs assessment, in the opinion of many systems analysts, is the most important part of the systems development process. No matter how well developed, even the best information system is useless if it doesn't meet the needs of its organization. A system that is not used represents wasted money.

A systems analyst has many tools and techniques available to help identify the needs of a new or modified information system:

- Observation: The systems analyst observes employees without interference. This allows users to demonstrate how they actually use the current system (be it automated or manual).
- Interviews: The systems analyst interviews employees at various levels in the organizational hierarchy. This process allows employees to communicate what works well with the current system and what needs to be changed. During the interviews, the analyst attempts to identify the differences among the perceptions of managers and those who work for them.

Sometimes a systems analyst will discover that what actually occurs is not what is supposed to be standard operating procedure. If there is a difference between what is occurring and the way in which things "should" happen, then either employee behavior will need to change or procedures will need to change to match employee behavior. It is not the systems analyst's job to make the choice, but only to document what is occurring and to present alternative solutions.

Occasionally observations and interviews can expose informal processes that may or may not be relevant to a new system. Consider what happened to a systems analyst who was working on a team that was developing an automated system for a large metropolitan library system. (This story is based on an incident that actually occurred in the 1980s.) The analyst was assigned to interview staff of the mobile services branch, the group that provided bookmobiles as well as individualized service to home-bound patrons. The process in question was book selection and ordering.

Here is how it was supposed to happen. Each week, the branch received a copy of a publication called *Publishers Weekly*. This magazine, which is still available, not only documents the publishing trade, but also lists and reviews forthcoming media (at the time, primarily books). The librarians (four adult librarians and one children's librarian) were to go through the magazine and place a checkmark by each book the branch should order. Once a month, the branch librarian was to take the marked up magazine to the central order meeting with all the other branch librarians in the system. All books with three or more checks should be ordered, although the branch librarian was to exercise her own judgment and knowledge of the branch patrons to help make appropriate choices.

and knowledge of the branch patrons to help make appropriate choices.

The preceding is what the systems analyst heard from the branch librarian. The five librarians, however, told a different story. At one point, they concluded that the branch librarian wasn't exercising any judgment at all but was simply ordering all books with three checks. There was only one children's librarian and therefore children's books almost never received three checks. Few children's books were being ordered.

To test their theory, the librarians placed four checkmarks next to a significantly inappropriate title—a coffee table book that was too heavy for many of their elderly patrons and that exceeded the branch's price limit—and waited to see what would happen. The coffee table book was ordered. It was clear to them that no professional judgment was being used at the order meeting.

The librarians, therefore, took the situation into their own hands. When the branch librarian returned from the order meeting, she gave the copy of the *Publishers Weekly* to one of the clerks, who created cards for each ordered book. The cards were to be matched to the new books when they arrived. However, the librarians arranged for the clerk to let them see the cards as soon as they were made. The librarians removed books that shouldn't be ordered and added those that had been omitted (primarily, children's books). The clerk then phoned the changes to the order department.

What should the analyst have done? The process was clearly broken. The branch librarian wasn't doing her job. The librarians had arranged things so that the branch functioned well, but they were circumventing standard operating procedure (SOP) and were probably placing their jobs in danger by doing so. This was a case of "the end justifies the means." No one was being harmed and the branch patrons were being helped. How should the analyst have reported her findings? Should she have exposed what was really happening or should she simply have documented how the procedure was *supposed* to work? What would happen when the ordering process was automated and there were no longer any centralized order meetings? There would be no order cards held at the branch and no opportunity for the librarians to correct the book order.

This was a very delicate situation because if it were exposed either the branch librarian and/or the other librarians would face significant problems. A systems analyst's job is to observe, interview, and record, not to intervene in employee relations. The book ordering process would be changing anyway with an automated system. If the librarians were to need to continue to work around the branch librarian, they would need to change their informal process as well. Therefore, the best strategy for the analyst probably was to remove herself from the personnel problems and report the process as it was supposed to work.

In other cases, where informal procedures do not violate SOP, an analyst can feel free to report what is actually occurring. This will help in tailoring the new information system to the way in which the business actually operates.

- Questionnaires: The systems analyst prepares questionnaires to be completed by employees. Like interviews, questionnaires give the systems analyst information about what is working currently and what needs to be fixed. The drawback to questionnaires, however, is that they are limited by the questions asked—even if they include open-ended questions—and may miss important elements of what a system needs to do.
- Focus groups: The systems analyst conducts a focus group to hear from employees who don't use the system directly and those who may not be employees, but who use the system in some way. For example, accountants may not use the payroll system directly, but may receive output from that system as input to the accounting system. A focus group can give them a forum to express how well the input meets their needs and how it might be changed to better suit them. A retail firm that sells through a Web site may conduct a focus group for customers to find out what changes should be made to the catalog and shopping cart interfaces.

Focus groups can be a double-edged sword. The members of the focus group are not privy to many of the constraints under which a business operates and the limitations of technology. Their suggestions may be impractical. The analyst conducting the focus session needs to be careful not to make promises to group members that can't be kept. Participants in a focus group can have their expectations raised so high that those expectations can never be met, creating disappointment and disaffection with the business.

- Brainstorming sessions: When employees know that something is not right with the current system, but are unable to articulate how it should be changed, a brainstorming session allows people to toss about ideas that may or may not be feasible. The intent is to stimulate everyone's thinking about the needs of a new or modified system without being critical.

The results of the needs assessment are collected into a *requirements document* or a requirements database. At this point in the process, the needs identified by the analyst are expressed in general terms. For example, the requirements document might include a need for a new Web site shopping cart that allowed users to check out on one page rather than three. The fact that the redesigned Web page needs to include the customer's area code as a separate piece of data is not documented at this point.

Assessing Feasibility

Once the operational requirements of a new system have been documented, the systems analyst turns to assessing the feasibility of the required system. There are three types of feasibility that systems analysts consider:

- Operational feasibility: Is it possible to develop a system that will fit within the organization's way of doing business or are any business process changes required realistic?
- Technical feasibility: Does the technology exist to implement a system to meet the organization's needs? (This includes not only the technology to implement the system, but the technology to provide adequate security.)
- Financial feasibility: Are the costs of implementing the system in a range that the organization is able/willing to pay?

Operational feasibility looks at whether it makes sense for the company to change any part of its operating procedures to accommodate a new system. If payroll employees enter data currently from time cards by hand, a change to a machine-readable system is operationally feasible. The procedure for entering hours worked changes slightly, but the employees will still use some sort of time card, and payroll employees will still be processing the data. In contrast, consider the situation where a new system would require all purchase requisitions to be placed using an online form. However, some offices in remote locations do not have consistent Internet access. It is therefore not feasible to shut down paper-based requisitions entirely.

Operational feasibility also relies to a large extent on an organization's and its employees' willingness to change. Assume, for example, that insurance company representatives fill out currently paper forms when making a sale at a customer's home. The company would like to replace the paper with laptops or tablets to exchange data with the home office. Certainly, this is a reasonable choice given the way in which the company operates, but if many of the salespeople are resistant to working with the mobile hardware, then a project to introduce the new devices may fail. Sometimes the introduction of new technology, especially in a manufacturing environment, is crucial to maintaining a company's competitive position and therefore its ability to stay in business. Employees who are resistant to the new technology (either unwilling or unable to be retrained) may need to be laid off to ensure that the company survives.

Technological feasibility is relatively easy to assess. Can the necessary technology be purchased? If not, is it possible to develop that technology in a reasonable amount of time and at a reasonable cost? Consider, for example, the situation of a rural library cooperative in the mid-1980s. Most library information systems used minicomputers. However, the rural cooperative was interested in a client-server architecture, with small servers at each library. Before proceeding, the cooperative needed to determine whether such a system actually existed or whether one was soon to

at each library. Before proceeding, the cooperative needed to determine whether such a system actually existed or whether one was soon to become available.[1]

Financial feasibility means asking the question: "Can we afford it?" Often the answer is "We can't afford not to do this." In such a situation, an organization will spend as much as it can to implement a new information system. To assess financial feasibility, an organization will undertake some type of cost/benefit analysis to answer the question "Do the benefits justify the costs?".

Because no specific system alternative has been selected at this point, financial feasibility assessment is often very general. It can be conducted in-house or an outside firm can be hired to conduct the analysis. The analysis includes market research to describe the market, its size, and typical customers as well as competition in the marketplace. From those data, the analyst can estimate demand for the company's product and generate an estimate of revenue. In addition to hardware and software costs, the cost estimates include facility expenses (rental, construction, and so on), financing (loan costs), and personnel expenses (hiring, training, salaries, and so on). The result is a projection of how a new information system will affect the bottom line of the company. As with the needs assessment, the results of the feasibility analysis are presented as part of the requirements document.

One thing to keep in mind during a feasibility analysis is that the systems analyst—whether an employee of the company contemplating a new system or an employee of an outside firm hired specifically to conduct the analysis—will not be making the decision as to whether an entire project will proceed. The decision is made by the organization's management.

Generating Alternatives

The third section in the system requirements document is a list of two or more system design alternatives for an organization to consider. Often they will be distinguished by cost (low cost, medium cost, and high cost). However, the first alternative is almost always "do nothing": Keep the current system.

A low-cost alternative generally takes advantage of as much existing facilities, hardware, and software as possible. It may rely more on changing employee behavior than installing new technology.

A moderate-cost alternative includes some new hardware and software purchases, some network modification, and some changes of employee behavior. It may not, however, include the top-of-the-line hardware or software. It may also use off-the-shelf software packages rather than custom-programmed applications. Employee training may include sending users to take classes offered by hardware and software vendors, either in person or online.

The high-cost solution usually includes the best of everything. Hardware and software are top-of-the-line and include a significant amount of excess capacity. Custom-programming is included where appropriate. Employee training includes on-site seminars tailored specifically to the organization.

Labor is an ongoing cost. Regardless of the alternative, the cost of personnel to keep the system running over time needs to be included. Labor costs, however, continue to increase and therefore are often underestimated.

Evaluating and Choosing an Alternative

Evaluating alternatives involves assigning numeric values for costs and benefits to of each alternative. Some costs are easy to quantify, especially the cost of hardware and prepackaged software. Labor can be estimated relatively easily as well (although as noted earlier, costs may rise significantly over time).

However, assigning dollar values to the benefits of a systems development project can be difficult because they are often intangible:

- When a system is being designed to generate an increase in sales volume, the amount of increase is by its very nature an estimate.
- Increased customer satisfaction and better employee attitudes are difficult to measure.
- Impact on system maintenance costs and future system development costs are at best an estimate.

Note: Doing nothing may not be the cost-free alternative that it at first appears to be. When doing nothing means losing customers because they can't order online, then doing nothing has a negative cost.

The analyst completes the requirements document by adding the cost/benefit analyses of the proposed alternatives and then presents it to company management. Although the requirements document typically includes specific groups of hardware, software, and labor for each alternative, management may decide to use part of one alternative (for example, existing hardware) and part of another (for example, custom-written application programs).

Once company management agrees to the project and its specifications, the requirements document can become a contract between IT and management,

defining what IT will provide and what management should expect. The more seriously all parties view the document as a contract, the more likely the system development project is to succeed.

Creating Design Requirements

The alternative chosen by an organization is usually expressed as a general strategy such as "implement a new Web site backed by an inventory database." Although many of the exact requirements of the database were collected during the systems analysis phase of the life cycle, company management usually doesn't really care about the details of which specific data will be in the database. The system that they chose has been specified as a series of outputs, the details of which may not have been designed as of yet.

Therefore, the first job in the design phase is to document exactly what data should be in the database and the details of application programs. This is the time when user interfaces are designed and an organization begins to build its data dictionary. Once the data specifications are in place, actual database design can begin.

Alternative Analysis Methods

As mentioned at the beginning of this chapter, the structured design life cycle works best when the requirements of an information system can be specified before development of the system begins. However, that is not always possible. Users may be able to express a general need—such as "It is imperative that we speed up the order entry process"— but are unable to articulate exactly what they want to see. In that situation, users may need the system developer to produce something to which they can react. They may not be able to articulate their needs in detail, but can indicate whether an existing piece of software works for them and if it does not, how the software should be changed to better meet their needs.

Prototyping

Prototyping is a form of systems development that is appropriate particularly in many situations where the exact requirements of an information system are not known in advance. Often the potential users know that help is needed, but can't articulate exactly what they want. The developer therefore begins by creating a shell for the system, consisting of user interface components but not necessarily any of the programs or databases behind them. This is the *prototype*.

The developer shows the users the prototype and lets them react to it. Based on user comments, the developer refines the prototype. The next version shown to the users may include user interface changes and some of the background programming.

The process can be summarized as:

1. Get a general idea of what the new information system should do.
2. Create a prototype.
3. Let the users react to the prototype.
4. Refine the prototype based on user input.
5. Return to step 3.
6. Repeat as many times as necessary until the users are satisfied.

A prototype may be missing some of the features of the final system. For example, initial programming may not include some of the security features or integrity controls that are likely to appear in the production product.

A prototype may be enhanced until it becomes the final system (*evolutionary prototyping*). In contrast, the prototype may be discarded once system requirements have been determined and the final system developed from scratch (*throwaway prototyping*). The latter is particularly useful when the code underlying the prototype has become difficult to follow (and thus maintain) because of the many changes that have been made during the prototyping process. Throwaway prototyping is also a very fast form of system development because it doesn't have to be "clean."

There are several drawbacks to prototyping. First, users may become confused between a prototype and a production system. They may expect the prototype to be a functioning whole and are therefore frustrated and disappointed when it is not. Second, prototyping doesn't include an analysis phase and relies solely on interaction between the users and the system developers to identify system requirements. Requirements that management may want to add to the system may not be included; users may leave out necessary system functions. Finally, prototyping may be expensive if the costs for developing the prototype are not controlled.

Database designers are usually involved after the first prototype is developed, and users have responded to it. The database design is created to provide whatever is needed to generate the outputs in the prototype and changes as the prototype is refined. The flexibility of relational database design is an enormous help to this methodology because of the ease in modifying the database structure.

Note: You will see an example of prototyping used in the case study in Chapter 14.

Spiral Methodology

The *spiral methodology* of systems analysis and design, which employs prototyping in a more formal way than the prototyping method, uses a gradual process in which each cycle further refines the system, bringing it closer to the desired end point. As you can see in Table 2.1, the methodology has four broad stages, each of which represents one trip around the spiral. The same type of activities is performed in each quadrant during each cycle. As you examine the table, also look at Figure 2.3. The activity numbers in the table correspond to the numbers on the spiral in the illustration.

Table 2.1

The Steps in the Spiral Systems Analysis and Design Methodology

Cycle	Quadrant	Specific Activities
Cycle 1	Quadrant 1: Plan next phases Quadrant 2: Determine objectives, abstractions, and constraints Quadrant 3: Evaluate alternatives; identify, resolve risks Quadrant 4: Design, verify next-level product	1. Requirements plan; life cycle plan 2. Risk analysis 3. Prototype #1 4. Concept of operation

Cycle 2	Quadrant 1: Plan next phases Quadrant 2: Determine objectives, abstractions, and constraints Quadrant 3: Evaluate alternatives; identify, resolve risks Quadrant 4: Design, verify next-level product	5. Development plan 6. Risk analysis 7. Prototype #2 8. Simulations 9. System requirements 10. Requirements validation
Cycle 3	Quadrant 1: Plan next phases Quadrant 2: Determine objectives, abstractions, and constraints Quadrant 3: Evaluate alternatives; identify, resolve risks Quadrant 4: Design, verify next-level product	11. Integration and test plan 12. Risk analysis 13. Prototype #3 14. Models 15. System design 16. Design validation and verification
Cycle 4	Quadrant 1: Plan next phases Quadrant 2: Determine objectives, abstractions, and constraints Quadrant 3: Evaluate alternatives; identify, resolve risks Quadrant 4: Design, verify next-level product	17. Determine process objectives, alternatives, and constraints. 18. Evaluate process alternatives, modify, resolve product risks 19. Design, verify next-level process plans 20. Risk analysis 21. Operational prototype 22. Benchmarks 23. Detailed design 24. Code 25. Unit test 26. Integration and test 27. Acceptance test 28. Implementation

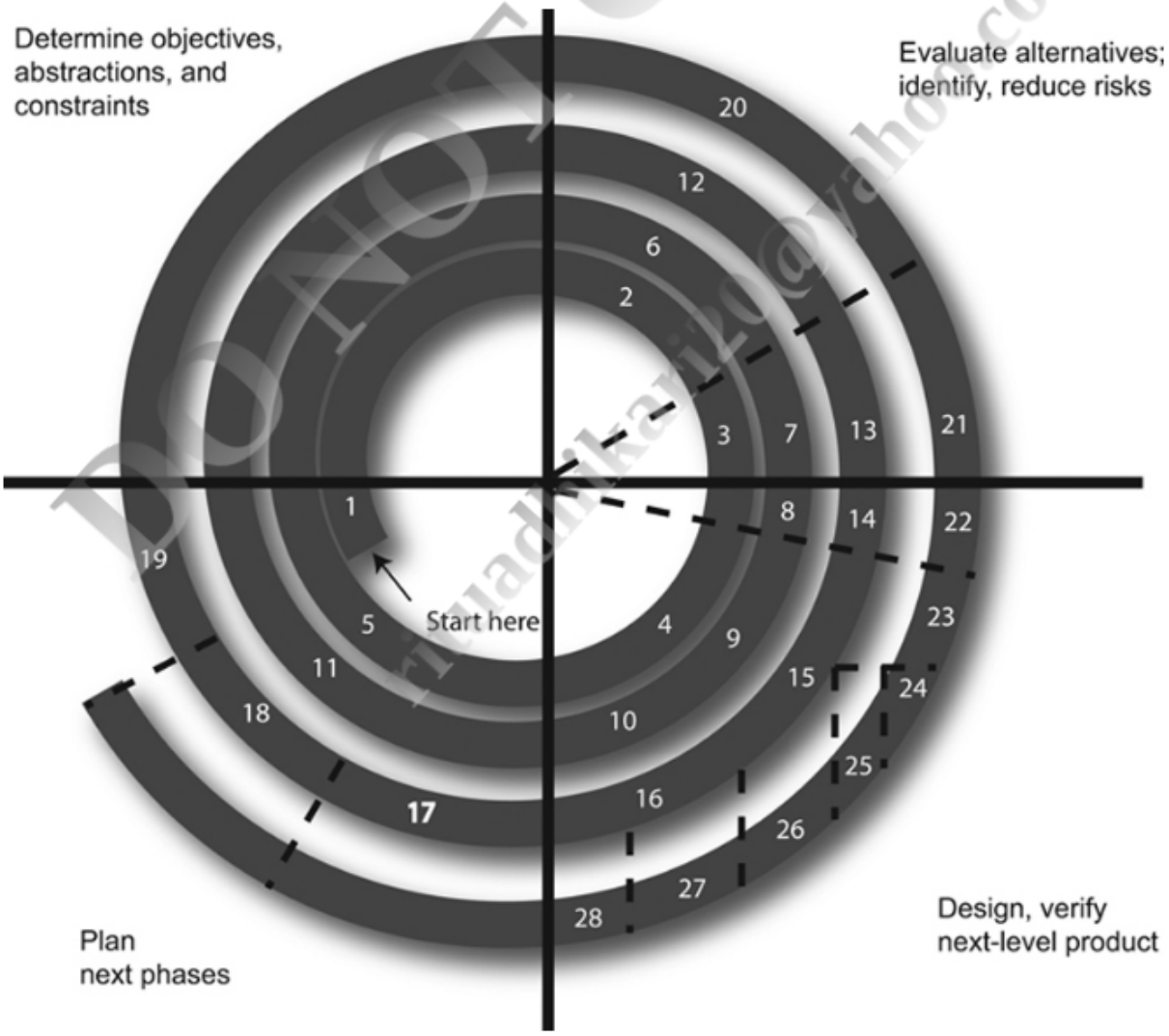

FIGURE 2.3 **The spiral systems analysis and design methodology (Numbers refer to Table 2.1).**

Notice that there are no specific activities listed for Quadrant 2 in any of the cycles. Systems analysis occurs in this quadrant, using the same techniques that are used to gather information for needs assessment in the traditional systems life cycle.

The spiral model is intended to address a perceived failing in the traditional system design cycle: analysis of the risk of the project. Although the traditional systems life cycle includes a feasibility analysis, there is no specific provision for looking at the chances that the system development project will succeed.

project will succeed.

Because there are prototype systems created during every cycle, database designers may be involved from throughout the entire process, depending on the characteristics of each prototype. The flexibility of a database design to change during the iterative development process becomes essential so that the database can be refined just as the other parts of the new system.

Object-Oriented Analysis

Object-oriented analysis is a method for viewing the interaction of data and manipulations of data that is based on the object-oriented programming paradigm. The traditional systems lifecycle looks at the outputs the system requires and then assembles the database so that it contains the data needed to produce those outputs. Documentation reflects the "input–process–output" approach, such that the inputs are identified to achieve a specified output; the process of translating the inputs into desired outputs is where the database and the programs that manipulate the database are found. Where the traditional systems analysis life cycle looks at data and data manipulation as two distinct parts of the system, object-oriented analysis focuses on units (*classes*) that combine data and procedures.

Although a complete discussion of object-oriented analysis and design is well beyond the scope of this book, a small example might help to make the distinction between traditional and object-oriented analysis clearer.[2] Assume that you are developing a system that will provide an employee directory of a company. Both forms of analysis begin with the requirements, such as the ability to search for employees in the directory and the ability to update employee information. Then the two approaches diverge.

Traditional analysis indicates the specifics of the application programs that will provide the updating and searching capabilities. The database specifications include the data needed to support those applications. The requirements document might contain something like Figure 2.4. The database itself would not have been designed. When it is, an employee might be represented something like Figure 2.5.

```
%NAME prepare_online_directory
%DEFINITION
Format and display an electronic version of the employee directory

%NAME prepare_print_directory
%DEFINITION
Format a directory for hard copy output

%NAME search_for_employee
%DEFINITION
Find an employee using the format "Last name, first name"

%NAME update_employee_information
%DEFINITION
Insert, modify, and delete employee information
```

FIGURE 2.4 Requirements for the employee directory.

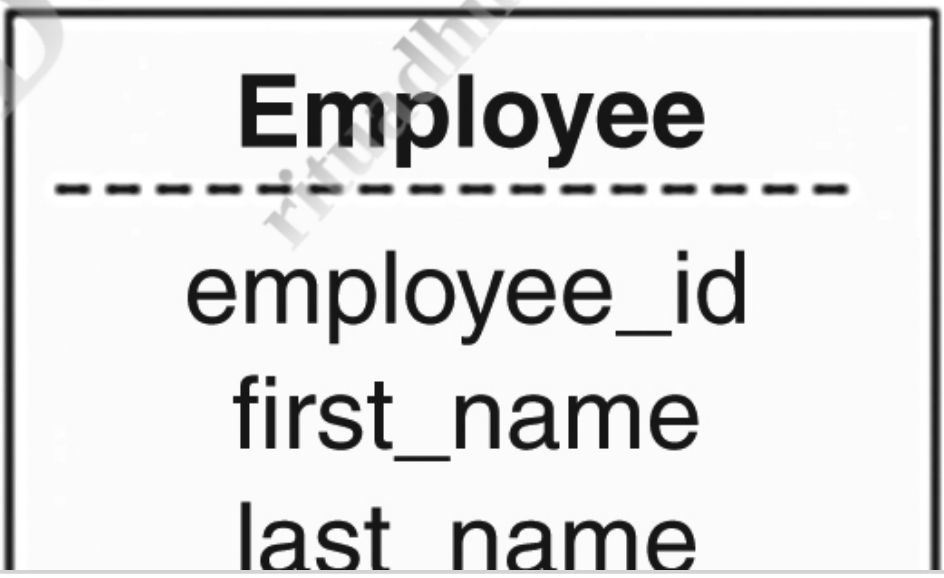

FIGURE 2.5 **A graphic representation of the data describing an employee for the employee directory.**

In contrast, object-oriented analysis specifies both the data and procedures together. In Figure 2.6 you will find a graphic representation of the employee directory environment. The rectangle labeled Employee is a class representing a single employee. The middle section of the rectangle contains the data that describe an employee for the directory. The bottom portion lists the things that an employee *object* (an instance of the class containing actual data) knows how to do.

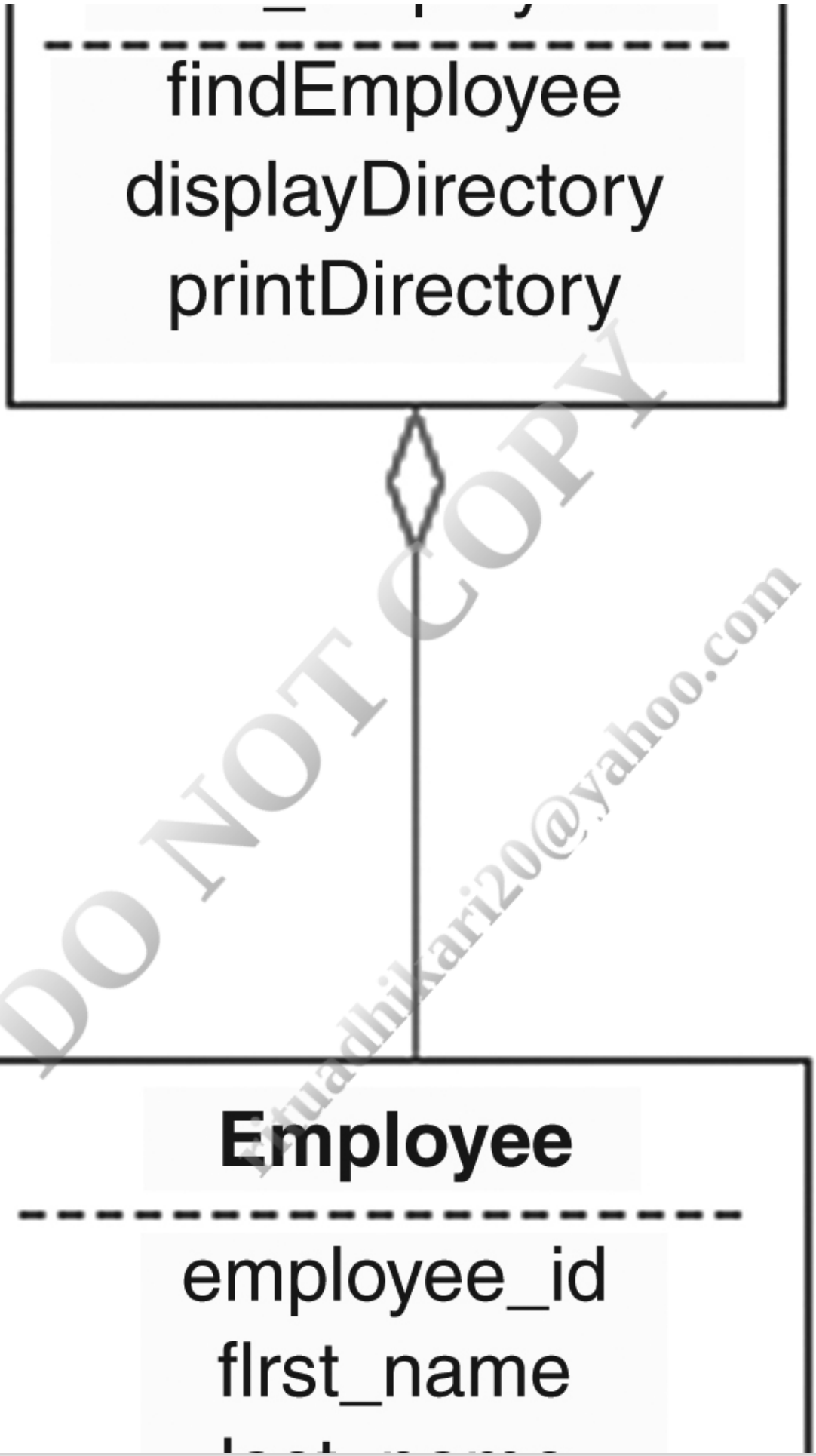

FIGURE 2.6 **A graphic representation of the object-oriented approach to the employee directory.**

The rectangle labeled AllEmployees is a class that gathers all the employee objects together (an *aggregation*). Procedures that are to be performed on all of the employees are part of this class. Diagrams of this type form the basis of an object-oriented requirements document and system design.

Object-oriented analysis is well-suited for large projects, especially those where requirements are likely to change as the system is being designed and developed. As object-oriented software development has replaced structured programming, object-oriented systems analysis has become increasingly popular.

For Further Reading

Arnowitz J, Arent M, Berger N. *Effective Prototyping for Software Makers*. Morgan Kaufmann; 2006.
Boehm, B., 1998. A Spiral Model of Software Development and Enhancement. IEEE Computer 21, 5, 61–72.
Clegg B, Birch P. *Instant Creativity: Techniques to Ignite Innovation & Problem Solving*. Kogan Page; 2007.
Dennis A, Wixom BH, Roth RM. *Systems Analysis and Design*. sixth ed. Wiley; 2014.
Hiatt J, Creasey T. *Change Management: The People Side of Change*. second ed. Prosci Learning Center Publications; 2012.
Hoffer JA, George J, VAlacich JA. *Modern Systems Analysis and Design*. seventh ed. Prentice Hall; 2013.
Kendall KE, Kendall JE. *Systems Analysis and Design*. ninth ed. Prentice Hall; 2013.
Kock NF. *Systems Analysis & Design Fundamentals: A business Process Redesign Approach*. Sage Publications; 2006.
Kotter JP. Harvard Business Review *HBR's 10 Must Reads on Change Management*. Harvard Business Review Press; 2011.
Krueger RA. *Focus Groups: A Practical Guide for Applied Research*. fourth ed. Sage Publications; 2008.
Luecke R. *Managing Change and Transition*. Harvard Business School Press; 2003.
Shoval P. *Functional and Object Oriented Analysis and Design: An Integrated Methodology*. IGI Global; 2006.
Whitten J. *Systems Analysis and Design Methods*. seventh ed. McGraw-Hill/Irwin; 2005.

[1] The final decision was to become beta-testers for the first library client-server hardware/software combination. Although the cooperative did have to deal with some bugs in the system, they paid less than would have otherwise.

[2] Details of the object-oriented paradigm and its role in relational databases can be found in Chapter 27.