IS231

Systems Analysis and Design

Prepared by

Dr. Samah Ahmed Zaki Hassan

Index

CHAPTER 4: THE ANALYSIS PROCESS

CHAPTER 5: UML CONCEPTS AND DIAGRAMS

CHAPTER 1
SYSTEMS, ROLES, AND BASIC CONCEPTS

1.1 Need for Systems Analysis and Design

The key to success in business is the ability to gather, organize, and interpret information. ***Systems Analysis and Design (SAD)*** is a proven methodology that helps both large and small businesses reap the rewards of utilizing information to its full capacity. It an exciting, active field in which analysts continually learn new techniques and approaches to develop systems more effectively and efficiently. All information systems projects move through the four phases of ***planning, analysis, design,*** and ***implementation***; all projects require analysts to gather requirements, model the business needs, and create blueprints for how the system should be built; and all projects require an understanding of organizational behavior concepts like change management and team building.

The major goal of systems analysis and design is to ***improve organizational systems***, typically through applying software that can help employees accomplish key business tasks more easily and efficiently. Often this process involves developing or acquiring application software and training employees to use it. Application software, also called a system, is designed to support a specific organizational function or process, such as inventory management, payroll, or market analysis. The goal of application software is to turn data into information. For example, software developed for the inventory department at a bookstore may keep track of the number of books in stock of the latest best seller. Software for the payroll department may keep track of the changing pay rates of employees. A variety of off-the-shelf application software can be purchased, including WordPerfect, Excel, and PowerPoint. ***off-the-shelf software*** are standardized software applications that are mass-produced/ready-made, available to the general public, and fit for immediate use. They are designed for a broad range of customers, offering a comprehensive set of features to streamline operations. However, off-the-shelf software may not fit the needs of a particular organization, and so the organization must develop its own product.

1.2 Roles of the System Analyst

As a systems analyst, the person in the organization most involved with systems analysis and design, you will enjoy a rich career path that will enhance both your computer and interpersonal skills. However, there is a core set of skills that all analysts need to know no matter what approach or methodology

is used. As a systems analyst, you will be at the center of developing this software. The analysis and design of information systems are based on the systems analyst:

- Understanding of the organization's objectives, structure, and processes.
- Knowledge of how to exploit information technology for advantage.

The systems analyst systematically assesses how users interact with technology and businesses function by examining the inputting and processing of data and the outputting of information with the intent of improving organizational processes. The analyst must be able to work with people of all descriptions and be experienced in working with computers. The analyst plays many roles, sometimes balancing several at the same time. The three primary roles of the systems analyst are ***consultant***, ***supporting expert***, and ***agent of change***.

1.2.1 Systems Analyst as Consultant

The systems analyst frequently acts as a systems consultant to humans and their businesses and, thus, may be hired specifically to ***address information systems issues*** within a business. Such hiring can be an advantage because outside consultants can bring with them a fresh perspective that other people in an organization do not possess. It also means that outside analysts are at a disadvantage because an outsider can never know the true organizational culture. As an outside consultant, you will rely heavily on the systematic methods to analyze and design appropriate information systems for users working in a particular business. In addition, you will rely on information systems users to help you understand the organizational culture from others'viewpoints.

1.2.2 Systems Analyst as Supporting Expert

In this role the analyst draws on professional expertise concerning computer hardware and software and their uses in the business. This work is often not a full-blown systems project, but rather it entails a small modification or decision affecting a single department.

As the supporting expert, you are not managing the project; you are merely serving as a resource for those who are. If you are a systems analyst employed by a manufacturing or service organization, many of your daily activities may be encompassed by this role.

1.2.3 Systems Analyst as Agent of Change

The most comprehensive and responsible role that the systems analyst takes on is that of an ***agent of change***, whether internal or external to the business. As an analyst, you are an agent of change whenever you perform any of the activities in the systems development life cycle (discussed in the next section) and are present and interacting with users and the business for an extended period (from two weeks to more than a year). An agent of change can be defined as a person who serves as a catalyst for change, develops a plan for change, and works with others in facilitating that change.

Your presence in the business changes it. As a systems analyst, you must recognize this fact and use it as a starting point for your analysis. Hence, you must interact with users and management from the very beginning of your project. Without their help you cannot understand what they need to support their work in the organization.

If change (that is, improvements to the business), the next step is to develop a plan for change along with the people who must enact the change. Once a consensus is reached on the change that is to be made, you must constantly interact with those who are changing.

As a systems analyst acting as an agent of change, you advocate a particular avenue of change involving the use of information systems. You also teach users the process of change, because changes in the information system do not occur independently; rather, they cause changes in the rest of the organization as well.

1.2.4 Qualities of the Systems Analyst

From the foregoing descriptions of the roles the systems analyst plays, it is easy to see that the successful systems analyst must possess a wide range of qualities. Many different kinds of people are systems analysts, so any description is destined to fall short in some way. There are some qualities, however, that most systems analysts seem to display. Above all:

1- The analyst is a ***problem solver***. He or she is a person who views the analysis of problems as a challenge and who enjoys devising workable solutions. When necessary, the analyst must be able to systematically tackle the situation at hand through skillful application of tools, techniques, and experience.

2- The analyst must also be a ***communicator*** capable of relating meaningfully to other people over extended periods of time. Systems analysts need to be able to understand humans' needs in interacting with technology, and they need enough computer experience to program, to understand the capabilities of computers, to glean information requirements from users, and to communicate what is needed to programmers.
3- The analyst need to ***possess strong personal and professional ethics*** to help them shape their client relationships.
4- The systems analyst must be a ***self-disciplined, self-motivated*** individual who is able to manage and coordinate other people, as well as innumerable project resources.

1.3 Levels of Management

Management in organizations exists on three broad, horizontal levels, Each level carries its own responsibilities, and all work toward achieving organizational goals and objectives in their own ways. The three levels of management are, as shown in the following Figure 1.1:

1- Operational control.
2- Managerial planning and control (middle management).
3- Strategic management.

Figure 1.1: Levels of Management in organizations

There are sharp contrasts among the decision makers on many dimensions. For instance, strategic managers have multiple decision objectives, whereas operations managers have single ones. It is often difficult for high-level managers to identify problems, but it is easy for operations managers to do so. Strategic managers are faced with semistructured problems, whereas lowerlevel managers deal mostly with structured problems. The alternative solutions to a problem facing the strategic managers are often difficult to articulate, but the alternatives that operations managers work with are usually easy to enumerate. Strategic managers most often make one-time decisions, whereas the decisions made by operations managers tend to be repetitive.

1.3.1 Operational control

Operational control forms the bottom tier of three-tiered management. Operations managers make decisions using predetermined rules that have predictable outcomes when implemented correctly. They make decisions that affect implementation in work scheduling, inventory control, shipping, receiving, and control of processes such as production. Operations managers oversee the operating details of the organization.

1.3.2 Middle management

Middle management forms the second, or intermediate, tier of the three-tiered management system. Middle managers make short-term planning and control decisions about how resources may best be allocated to meet organizational objectives. Their decisions range all the way from forecasting future resource requirements to solving employee problems that threaten productivity. The decision-making domain of middle managers can usefully be characterized as partly operational and partly strategic, with constant fluctuations.

1.3.3 Strategic management

Strategic management is the third level of three-tiered management control. Strategic managers look outward from the organization to the future, making decisions that will guide middle and operations managers in the months and years ahead. Strategic managers work in a highly uncertain decision-making environment. Through statements of goals and the determination of strategies and policies to achieve them, strategic managers actually define the organization as a whole. Theirs is the broad picture, wherein the company decides to develop new product lines, divest itself of unprofitable ventures, acquire other compatible companies, or allow itself to be acquired or merged.

1.4 Types of Systems

Information systems are developed for different purposes, depending on the needs of human users and the business. Transaction processing systems (TPS) function at the operational level of the organization; office automation systems (OAS) and knowledge work systems (KWS) support work at the knowledge level. Higher-level systems include management information systems (MIS) and decision support systems (DSS). Expert systems apply the expertise of decision makers to solve specific, structured problems. On the strategic level of management we find executive support systems (ESS). Group decision support systems (GDSS) and the more generally described computer-supported collaborative work systems (CSCWS) aid group-level decision making of a semistructured or unstructured variety. The variety of information systems that analysts may develop is shown in the following Figure 1.2.

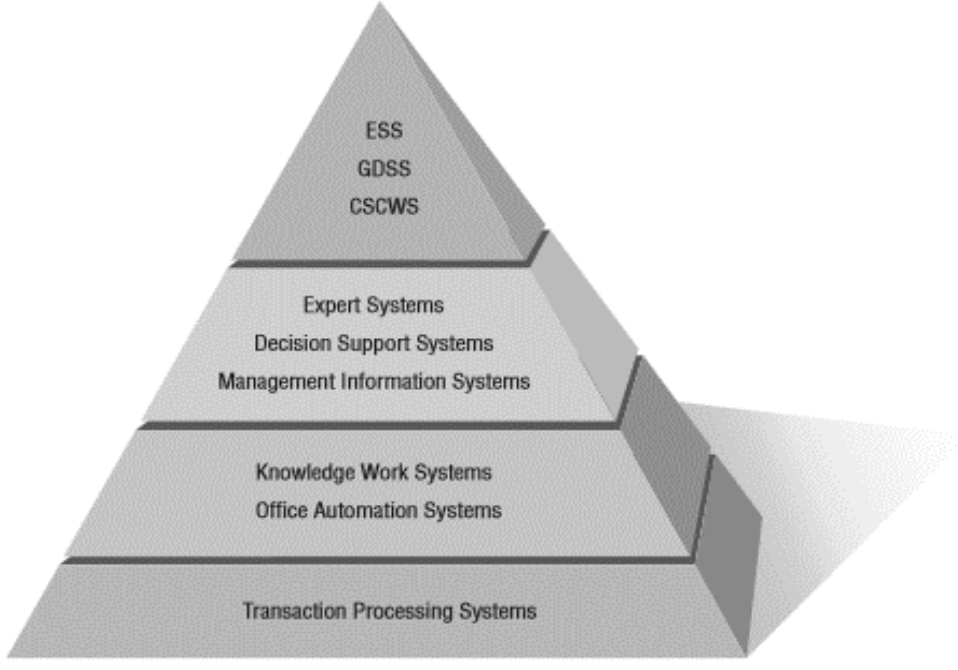


Figure 1.2: Types of Systems.

Notice that the figure presents these systems from the bottom up, indicating that the operational, or lowest, level of the organization is supported by TPS, and the strategic, or highest, level of semistructured and unstructured decisions is supported by ESS, GDSS, and CSCWS at the top. This text uses the terms management information systems, information systems (IS), computerized information systems, and computerized business information systems interchangeably to denote computerized information systems that support the broadest range of user interactions with technologies and business activities through the information they produce in organizational contexts.

1.5 Integrating Technologies for Systems

As users adopt new technologies, some of the systems analyst's work will be devoted to integrating traditional systems with new ones to ensure a useful context, as shown in Figure 1.3. This section describes some of the new information technologies systems analysts will be using as people work to integrate their ecommerce applications into their traditional businesses or as they begin entirely new ebusinesses.

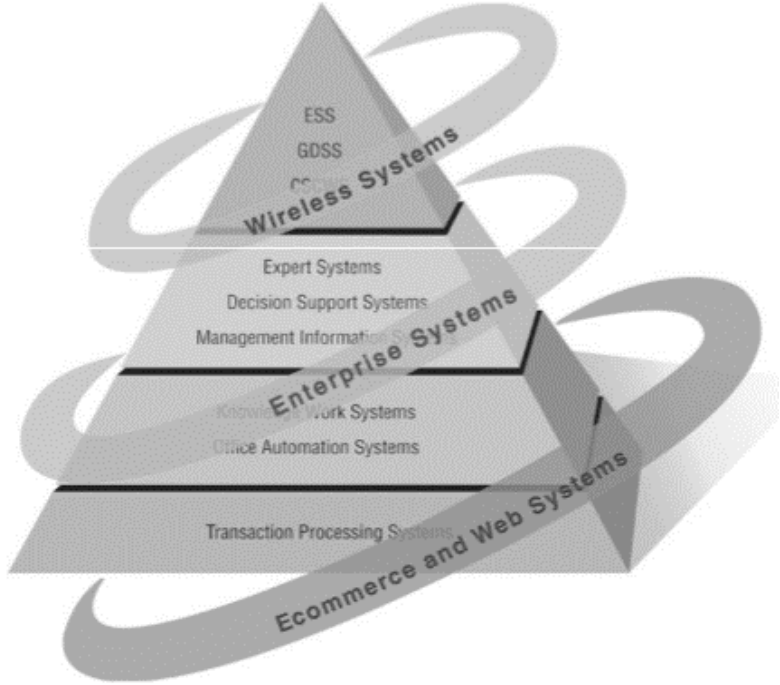


Figure 1.3: Integrating Technologies for Systems.

Systems analysts need to be aware that integrating technologies affect all types of users and systems.

1.5.1 Ecommerce Applications and Web Systems

Many of the systems discussed here can be imbued with greater functionality if they are migrated to the World Wide Web or if they are originally conceived and implemented as Web-based technologies. There are many benefits to mounting or improving an application on the Web:

1. Increasing user awareness of the availability of a service, product, industry, person, or group.
2. The possibility of 24-hour access for users.
3. Improving the usefulness and usability of the interface design.
4. Creating a system that can extend globally rather than remain local, thus reaching people in remote locations without worry of the time zone in which they are located.

1.5.2 Enterprise Systems

Many organizations envision potential benefits from the integration of many information systems existing on different management levels and within different functions. Some authors discuss integration as ***service-oriented architecture (SOA)***. Enterprise systems would comprise the top layer. Enterprise systems, also called ***enterprise resource planning (ERP)*** systems, are designed to perform this integration. Instituting ERP requires enormous commitment and organizational change. Often systems analysts serve as consultants to ERP endeavors that use proprietary software. Typically, analysts as well as some users require vendor training, support, and maintenance to be able to properly design, install, maintain, update, and use a particular ERP package.

1.5.3 Systems for Wireless and Mobile Devices

Analysts are being asked to design an increase of new systems and applications for adventurous users, including many for wireless and mobile devices such as the Apple iPhone, iPod, or the BlackBerry. In addition, analysts may find themselves designing standard or wireless communications networks for users that integrate voice, video, text messaging, and email into organizational intranets or industry extranets. Wireless ecommerce is referred to as ***mcommerce (mobile commerce)***.

Wireless local area networks (WLANs); wireless fidelity networks, called ***Wi-Fi***; and personal wireless networks that bring together many types of devices under the standard called Bluetooth are all systems that you may be asked to design. In more advanced settings, analysts may be called on to design ***intelligent agents***, software that can assist users with tasks in which the software learns users' preferences over time and then acts on those preferences. For example, in the use of ***pull technology***, an intelligent agent would search the Web for stories of interest to the user, having observed the user's behavior patterns with information over time, and would conduct searches on the Web without continual prompting from the user.

1.6 Human–Computer Interaction Considerations

In recent years, the study of human–computer interaction (HCI) has become increasingly important for systems analysts. Although the definition is still evolving, researchers characterize ***HCI*** as the "aspect of a computer that enables communications and interactions between humans and the computer. It is the layer of the computer that is between humans and the computer. Analysts using an HCI approach are emphasizing people rather than the work to be done or the IT that is involved. Their approach to a problem is multifaceted, looking at the "human ergonomic, cognitive, affective, and behavioral factors involved in user tasks, problem solving processes and interaction context". Human computer interaction moves away from focusing first on organizational and system needs and instead concentrates on human needs.

Analysts adopting HCI principles examine a variety of user needs in the context of humans interacting with information technology to complete tasks and solve problems. These include taking into account physical or ergonomic factors; usability factors that are often labeled cognitive matters; the pleasing, aesthetic, and enjoyable aspects of using the system; and behavioral aspects that center on the usefulness of the system.

Another way to think about HCI is to think of it as a ***human-centered approach*** that puts people ahead of organizational structure or culture when creating new systems. When analysts employ HCI as a lens to filter the world, their work will possess a different quality than the work of those who do not possess this perspective.

Your career can benefit from a strong grounding in HCI. The demand for analysts who are capable of incorporating HCI into the systems development

process keeps rising, as companies increasingly realize that the quality of systems and the quality of work life can both be improved by taking a human-centered approach at the outset of a project.

The application of human–computer interaction principles tries to uncover and address the frustrations that users voice over their use of information technology. These concerns include a suspicion that systems analysts misunderstand the work being done, the tasks involved, and how they can best be supported; a feeling of helplessness or lack of control when working with the system; intentional breaches of privacy; trouble navigating through system screens and menus; and a general mismatch between the system designed and the way users themselves think of their work processes.

Misjudgments and errors in design that cause users to neglect new systems or that cause systems to fall into disuse soon after their implementation can be eradicated or minimized when systems analysts adopt an HCI approach.

Researchers in HCI see advantages to the inclusion of HCI in every phase of the SDLC. This is a worthwhile approach, and we will try to mirror this by bringing human concerns explicitly into each phase of the SDLC. As a person who is learning systems analysis, you can also bring a fresh eye to the SDLC to identify opportunities for designers to address HCI concerns and ways for users to become more central to each phase of the SDLC.

CHAPTER 2
SYSTEMS ANALYSIS AND DESIGN METHODOLOGIES

2.1 The Systems Approach

Developing successful information system solutions to business problems is a major challenge for business managers and professionals today. As a business professional, you will be responsible either for proposing, assisting with development, or developing new or improved uses of information technologies for your company. As a business manager, you will frequently manage the development efforts of information systems specialists and other business end users.

A problem-solving process is called the ***systems approach***. When the systems approach to problem solving is applied to the development of information systems solutions to business problems, it is called ***information systems development*** or ***application development***.

The systems approach to problem solving uses a systems orientation to define problems and opportunities and then develop appropriate, feasible solutions in response. Analyzing a problem and formulating a solution involve the following interrelated activities:

1. Recognize and define a problem or opportunity using systems thinking.
2. Develop and evaluate alternative system solutions.
3. Select the system solution that best meets your requirements.
4. Design the selected system solution.
5. Implement and evaluate the success of the designed system.

The essence of the discipline of systems thinking is "seeing the forest and the trees" in any situation by:

- Seeing interrelationships among systems rather than linear cause-and-effect chains whenever events occur.
- Seeing processes of change among systems rather than discrete "snapshots" of change, whenever changes occur.

One way of practicing systems thinking is to try to find systems, subsystems, and components of systems in any situation you are studying. This is also known as using a systems context, or having a systemic view of a situation. For example, the business organization or business process in which a problem or opportunity arises could be viewed as a system of input, processing, output, feedback, and control components. Then to understand a problem and solve it, you would determine whether these basic systems functions are being properly performed.

Example.

The sales process of a business can be viewed as a system. You could then ask: Is poor sales performance (output) caused by inadequate selling effort (input), out-of-date sales procedures (processing), incorrect sales information (feedback), or inadequate sales management (control)? Figure 2.1 illustrates this concept, as we can better understand a sales problem or opportunity by identifying and evaluating the components of a sales system.

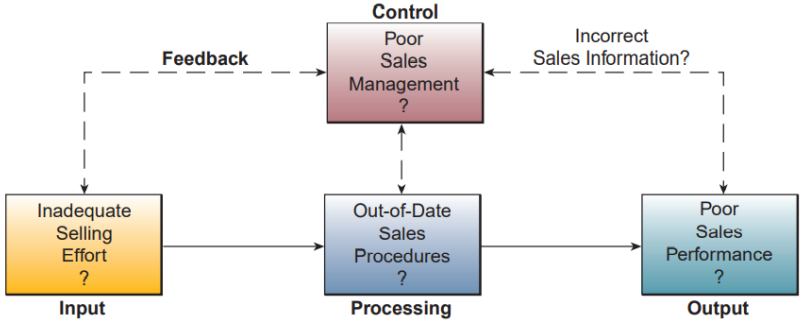


Figure 2.1: An example of systems thinking.

2.2 Project Management

Any discussion of information systems design and development would be incomplete without including a discussion of basic project management concepts, techniques, and tools. Before we progress any further in our discussion of implementation, we need to understand how our project, which we hope is on time and within budget, got to this point.

A ***project*** is a special set of activities with a clear beginning and end. Every project has a set of ***goals***, ***objectives***, and ***tasks***. Every project must also deal with a set of ***limitations*** or ***constraints***. Finally, although the content can vary from one project to the next, there are many important similarities in the process. The first, and probably the greatest, contribution of the modern project management approach is to identify the project as a series of steps or phases. The SDLC is a project management approach tailored toward the design and development of information systems. Before we return our focus to a specific project management approach such as the SDLC, let's look at a more generic picture of project management and see how it compares. No matter what the project, three elements will be necessary to manage it effectively and efficiently: ***process, tools ,*** and ***techniques***.

2.2.1 The Process of Project Management

The modern project management approach has identified five phases in the process, Table 2.1.

Table 2.1: The five phases of project management.

Project Management Phase	Example Activities
Initiating/Defining	• State the problem(s)/goal(s). • Identify the objectives. • Secure resources. • Explore costs/benefits in feasibility study.
Planning	• Identify and sequence activities. • Identify the "critical path." • Estimate time and resources needed for completion. • Write a detailed project plan.
Executing	• Commit resources to specific tasks. • Add additional resources/personnel if necessary. • Initiate project work.
Controlling	• Establish reporting obligations. • Create reporting tools. • Compare actual progress with baseline. • Initiate control interventions if necessary.
Closing	• Install all deliverables. • Finalize all obligations/commitments. • Meet with stakeholders. • Release project resources. • Document the project. • Issue final report.

2.2.2 Initiating and Defining

The first phase of the project management process serves as a foundation for all that follows. The most important objective to achieve during this phase is the clear and succinct statement of the ***problem*** that the project is to solve or the ***goals*** that the project is to achieve. Any ambiguity at this point often spells doom for even the best-executed projects. Also during this phase, it is necessary to ***identify and secure the resources*** necessary to execute the project, explore the costs and benefits, and identify any risks. As recognized,

this is exactly what happens during the systems ***investigation phase*** of the SDLC.

2.2.3 Planning

Here every project ***objective*** and every ***activity*** associated with that objective must be identified and sequenced. Several tools have been created to assist in the sequencing of these activities including *simple dependence diagrams, program evaluation and review (PERT), critical path method (CPM),* and a commonly used timeline diagram known as a *Gantt chart*. The common use of all of these tools is to help plan and sequence activities associated with the project objectives so that nothing is left out or done twice. These same tools also help the project manager determine how long each activity will take and, thus, how long the project will take. Later in the project process, the tools will help determine whether the project is on schedule and, if not, where the delays occurred and what can be done to remedy the delay.

2.2.4 Executing

Once all of the activities in the planning phase are complete and all detailed plans have been created and approved, the execution phase of the project can begin. It is here that all of the plans are put into motion. Resources, tasks, and schedules are brought together, and the necessary work teams are created and set forth on their assigned paths. In many respects, this is the most exciting part of the project management process. The phases of ***systems analysis and system design*** are the primary phases associated with project execution in the SDLC.

2.2.5 Controlling

Some project management experts suggest that controlling is just an integral part of the execution phase of project management; others suggest it must be viewed as a separate set of activities that, admittedly, occur simultaneous to the execution phase. In either case, it is important to give sufficient attention to the controlling activities to ensure that the project objectives and ***deadlines*** are met. Probably the single most important tool for project control is the ***report***. Three common types of reports are generated to assist with project control.

- The ***variance report*** contains information related to the difference between actual and planned project progress. It helps identify when a

project is off track but provides little evidence as to what is causing the delay.

- The ***status report*** is an open-ended report that details the process that led to the current project state. By analyzing this report, a project manager can pinpoint where the delay began and can create a plan to get past it and possibly make up for lost time.
- The ***resource allocation report*** identifies the various resources (people, equipment, and so on) that are being applied to specific project activities, as well as where currently unused, or slack , resources may be available.

The second and third types of reports are more helpful in determining the cause of delays and the appropriate corrections.

2.2.6 Closing

This last phase of the project management process focuses on bringing a project to a successful end. The beginning of the end of a project is the ***implementation and installation*** of all of the project deliverables. The next step is the ***formal release*** of the project resources so they can be redeployed into other projects or job roles. The final step in this phase is to review the ***final documentation*** and publish the ***final project report***. This is where the good and bad news concerning the project are documented, and the elements necessary for a postproject review are identified.

2.3 Systems Analysis and Design approaches

The overall process by which information systems are designed and implemented within organizations is referred to as ***systems analysis and design (SA&D)***. There are many approaches to SA&D. The two most common approaches are ***object-oriented analysis and design*** and the ***life cycle approach***. Although each has its advantages and disadvantages, and the two approaches differ in many respects, both are concerned with the analysis and design of a successful information system. In most cases, the choice will depend upon the type of system under study and the degree to which users are able to specify their needs and requirements clearly.

2.3.1 The System Development Life Cycle (SDLC)

One method of using the systematic approach analysts take to the analysis, design, and develop information system solutions, and the most prevalent one

in organization systems analysis and design, can be viewed as a multistep, iterative process called the ***systems development life cycle (SDLC)***. The SDLC is a phased approach to analysis and design that holds that systems are best developed through the use of a specific cycle of analyst and user activities. Analysts disagree on exactly how many phases there are in the SDLC, but they generally laud its organized approach.

Figure 2.2 shows that several major activities must be accomplished and managed in a complete IS development cycle.

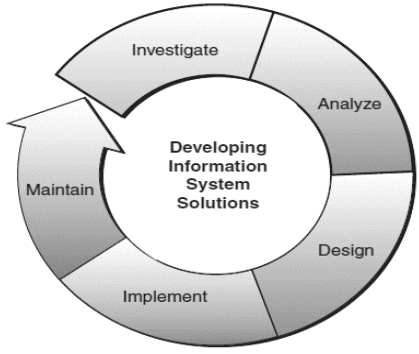


Figure 2.2: The system Developing Life Cycle (SDLC)

Figure 2.3 divided the cycle into seven phases. Although each phase is presented discretely, it is never accomplished as a separate step. Instead, several activities can occur simultaneously, and activities may be repeated.

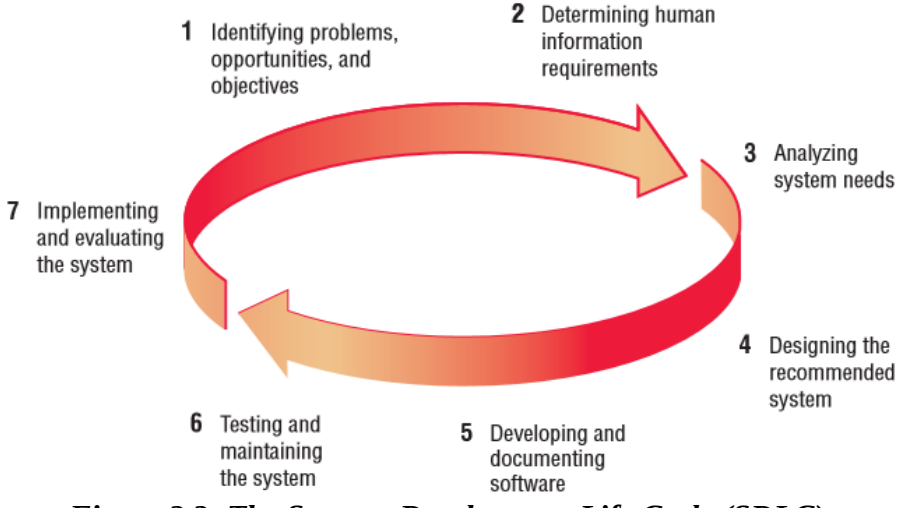


Figure 2.3: The Systems Development Life Cycle (SDLC).

From Figure 2.2 and Figure 2.3 we can get that the investigation phase may include the identifying problems, opportunities and objectives phase and the determining human information requirements phase. Also we can get that the implementation and maintenance phases in the five phases SDLC are as same as phases from 5 to 7 phases in the seven phases SDLC. The following subsections will address those phases in more details.

2.3.1.1 Systems Investigation Stage

This is the first step in the systems development process in which some questions have to be answered like: Do we have business opportunities? What are our business priorities? How can information technologies provide information system solutions that address our business priorities?

This stage may involve consideration of ***proposals*** generated by a business/IT planning process. The investigation stage also includes the preliminary ***feasibility study*** of proposed information system solutions to meet a company's business ***priorities*** and ***opportunities***.

The goal of the preliminary feasibility study is to evaluate alternative system solutions and to propose the most feasible and desirable business application for development. The feasibility of a proposed business system can be evaluated in terms of ***five major categories***, as illustrated in Table 2.2.

Table 2.2: **Feasibility Study**

Operational Feasibility	- How well the proposed system supports the business priorities of the organization. - How well the proposed system will solve the identified problem. - How well the proposed system will fit with the existing organizational structure.
Economic Feasibility	- Cost savings. - Increased revenue. - Decreased investment requirements. - Increased profits. - Cost/benefit analysis.
Technical Feasibility	- Hardware, software, and network capability, reliability, and availability.
Human Factors Feasibility	- Employee, customer, supplier acceptance. - Management support. - Determining the right people for the various new or revised roles.
Legal/Political Feasibility	- Patent, copyright, and licensing. - Governmental restrictions. - Affected stakeholders and reporting authority.

- **Identifying Problems, Opportunities, and Objectives**

In this phase/step, the analyst is concerned with correctly identifying problems, opportunities, and objectives. This stage is critical to the success of the rest of the project, because no one wants to waste subsequent time addressing the wrong problem.

The first phase requires that the analyst look honestly at what is occurring in a business. Then, together with other organizational members, the analyst pinpoints problems. Often others will bring up these problems, and they are the reason the analyst was initially called in. ***Opportunities*** are situations that the analyst believes can be improved through the use of computerized information systems. Seizing opportunities may allow the business to gain a competitive edge or set an industry standard.

Identifying ***objectives*** is also an important component of the first phase. The analyst must first discover what the business is trying to do. Then the analyst will be able to see whether some aspect of information systems applications can help the business reach its objectives by addressing specific problems or opportunities.

The ***people*** involved in the first phase are the users, analysts, and systems managers coordinating the project. ***Activities*** in this phase consist of interviewing user management, summarizing the knowledge obtained, estimating the scope of the project, and documenting the results. The ***output*** of this phase is a feasibility report containing a problem definition and summarizing the objectives. Management must then make a decision on whether to proceed with the proposed project. If the user group does not have sufficient funds in its budget or wishes to tackle unrelated problems, or if the problems do not require a computer system, a different solution may be recommended, and the systems project does not proceed any further.

- **Determining Human Information Requirements**

The next phase/step the analyst enters is that of determining the human needs of the users involved, using a variety of tools to understand ***how users interact*** in the work context with their current information systems. The analyst will use interactive methods such as ***interviewing***, ***sampling*** and ***investigating*** hard data, and ***questionnaires***, along with unobtrusive methods, such as observing decision makers' behavior and their office environments, and all-encompassing methods, such as prototyping.

The analyst will use these methods to pose and answer many questions concerning human-computer interaction (HCI), including questions such as, "What are the users' physical strengths and limitations?" In other words, "What needs to be done to make the system audible, legible, and safe?" "How can the new system be designed to be easy to use, learn, and remember?" "How can the system be made pleasing or even fun to use?" "How can the system support a user's individual work tasks and make them more productive in new ways?"

In the information requirements phase of the SDLC, the analyst is striving to understand what information users need to perform their jobs. At this point the analyst is examining how to make the system useful to the people involved. How can the system better support individual tasks that need doing? What new tasks are enabled by the new system that users were unable to do without it? How can the new system be created to extend a user's capabilities beyond what the old system provided? How can the analyst create a system that is rewarding for workers to use?

The ***people*** involved in this phase are the analysts and users, typically operations managers and operations workers. The systems analyst needs to know the details of current system functions: the who (the people who are involved), what (the business activity), where (the environment in which the work takes place), when (the timing), and how (how the current procedures are performed) of the business under study. The analyst must then ask why the business uses the current system. There may be good reasons for doing business using the current methods, and these should be considered when designing any new system.

If the reason for current operations is that "it's always been done that way," however, the analyst may wish to improve on the procedures. ***At the completion of this phase***, the analyst should understand how users accomplish their work when interacting with a computer and begin to know how to make the new system more useful and usable. The analyst should also know how the business functions and have complete information on the people, goals, data, and procedures involved.

2.3.1.2 Analyzing System Needs

The ne0xt phase that the systems analyst undertakes involves analyzing system needs. Again, special tools and techniques help the analyst make requirement determinations. Tools such as ***data flow***

diagrams (DFD) to chart the input, processes, and output of the business's functions, or ***activity diagrams or sequence diagrams*** to show the sequence of events, illustrate systems in a structured, graphical form. From data flow, sequence, or other diagrams, a ***data dictionary*** is developed that lists all the data items used in the system, as well as their specifications.

During this phase the systems analyst also analyzes the structured decisions made. ***Structured decisions*** are those for which the conditions, condition alternatives, actions, and action rules can be determined. There are three major methods for analysis of structured decisions: ***structured English, decision tables,*** and ***decision trees***.

At this point in the SDLC, the systems analyst prepares a systems proposal that summarizes what has been found out about the users, usability, and usefulness of current systems; provides ***cost-benefit analyses*** of alternatives; and makes recommendations on what (if anything) should be done. If one of the recommendations is acceptable to management, the analyst proceeds along that course. Each systems problem is unique, and there is never just one correct solution. The manner in which a recommendation or solution is formulated depends on the individual qualities and professional training of each analyst and the analyst's interaction with users in the context of their work environment.

Whether you want to develop a new application quickly or are involved in a long-term project, you will need to perform several basic activities of systems analysis. Many of these activities are an extension of those used in conducting a feasibility study. Systems analysis is not a preliminary study; however, it is an indepth study of end-user information needs that produces ***functional requirements*** that are used as the basis for the design of a new information system. Systems analysis traditionally involves a detailed study of:

- The information needs of a company and end users like yourself.
- The activities, resources, and products of one or more of the present information systems being used.
- The information system capabilities required to meet your information needs, and those of other business stakeholders that may use the system.

The following table shows some Examples of functional requirements for a proposed e-commerce system for a business.

Table 2.3: Examples of functional requirements

User Interface Requirements	Automatic entry of product data and easy-to-use data entry screens for Web customers.
Processing Requirements	Fast, automatic calculation of sales totals and shipping costs.
Storage Requirements	Fast retrieval and update of data from product, pricing, and customer databases.
Control Requirements	Signals for data entry errors and quick e-mail confirmation for customers.

2.3.1.3 Designing the Recommended System

Once the analysis portion of the life cycle is complete, the process of systems design can begin. Here is where the ***logical model*** of the current system is modified until it represents the ***blueprint*** for the new system. This version of the logical model represents what the new system will do. During the ***physical design*** portion of this step, users and analysts will focus on determining how the system will accomplish its objectives. This is where issues related to hardware, software, networking, data storage, security, and many others will be discussed and determined. As such, systems design consists of design activities that ultimately produce physical system specifications satisfying the functional requirements that were developed in the systems analysis process.

In the design phase of the SDLC, the systems analyst uses the information collected earlier to accomplish the ***logical design*** of the information system. The analyst ***designs procedures*** for users to help them accurately enter data so that data going into the information system are correct. In addition, the analyst provides for users to complete effective input to the information system by using techniques of good form and Web page or screen design.

Part of the ***logical design*** of the information system is devising the HCI. The interface connects the user with the system and is thus extremely important. The ***user interface*** is designed with the help of users to make sure that the system is audible, legible, and safe, as well as attractive and enjoyable to use. Examples of physical user interfaces include a keyboard (to type in questions and answers), onscreen menus (to elicit user commands), and a variety of ***graphical user interfaces (GUIs)*** that use a mouse or touch screen.

The design phase also includes ***designing databases*** that will store much of the data needed by decision makers in the organization. Users benefit from a well-organized database that is logical to them and corresponds to the way

they view their work. In this phase the analyst also works with users to ***design output*** (either onscreen or printed) that meets their information needs.

Finally, the analyst must ***design controls and backup procedures*** to protect the system and the data, and to produce program specification packets for programmers. Each packet should contain input and output layouts, file specifications, and processing details; it may also include decision trees or tables, UML or data flow diagrams, and the names and functions of any prewritten code that is either written in-house or using code or other class libraries.

A useful way to look at systems design is illustrated in Figure 2.4, as it can be viewed as the design of user interfaces, data, and processes.. This concept focuses on three major products, or ***deliverables***, that should result from the design stage. In this framework, systems design consists of three activities: ***user interface, data,*** and ***process design***. This results in specifications for user interface methods and products, database structures, and processing and control procedures.

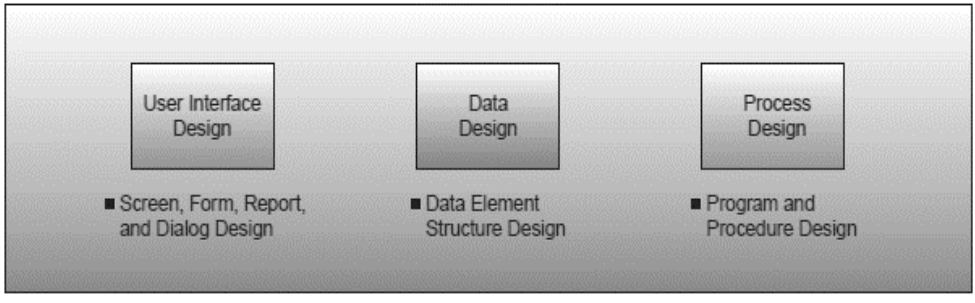


Figure 2.4: System Design

During the design phase, the development process frequently takes the form of, or includes, a ***prototyping approach***.

2.3.1.4 Implementation

Once a new information system has been designed, it must be implemented as a working system and maintained to keep it operating properly. The implementation process we will cover in this section follows the investigation, analysis, and design stages of the systems development life cycle. Implementation is a vital step in the deployment of information technology to support the employees, customers, and other business stakeholders of a company.

Figure 2.5 shows an overview of the ***implementation process***. Implementation activities are needed to transform a newly developed information system into an operational system for end users. It illustrates that the systems implementation stage involves hardware and software acquisition, software development, testing of programs and procedures, conversion of data resources, and a variety of conversion alternatives. It also involves the education and training of end users and specialists who will operate a new system.

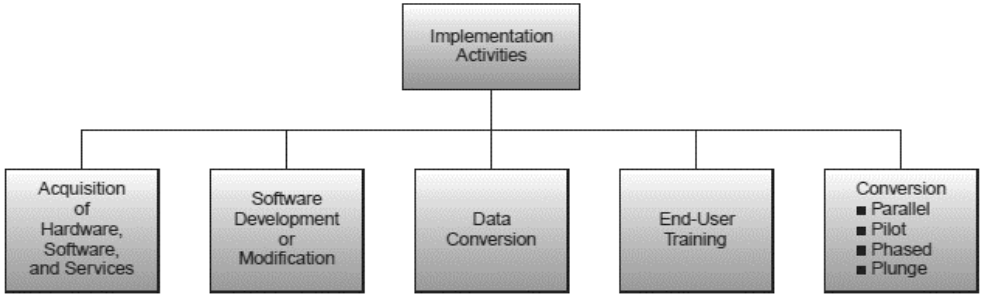


Figure 2.5: An overview of the implementation process.

Implementation can be a difficult and time-consuming process; however, it is vital in ensuring the success of any newly developed system. Even a well-designed system will fail if it is not properly implemented, which is why the implementation process typically requires a ***project management*** effort on the part of IT and business unit managers. They must enforce a project plan, which includes job responsibilities, timetables for major stages of development, and financial budgets. This is necessary if a project is to be completed on time and within its established budget, while still meeting its design objectives.

The following subsections discuss other implementation activities like Testing, data conversion, documentation, and training are keys to successful implementation of a new business system.

- **Evaluating Hardware, Software, and Services**

A major activity during the implementation phase of the SDLC is the acquisition of the hardware and software necessary to implement the new system. How do companies evaluate and select hardware, software, and IT services. Minimum acceptable physical and performance characteristics for all hardware and software requirements are established. Most large business firms and all government agencies formalize these requirements by listing them in a document called an ***request for proposal (RFP)*** or ***request for***

quotation(RFQ). Then they send the RFP or RFQ to appropriate vendors, who use it as the basis for preparing a proposed purchase agreement.

Companies may use a ***scoring*** system of evaluation when there are several competing proposals for a hardware or software acquisition. They give each ***evaluation factor*** a certain number of maximum possible points. Then they assign each competing proposal points for each factor, depending on how well it meets the user's specifications. Scoring evaluation factors for several proposals helps organize and document the evaluation process. It also spotlights the strengths and weaknesses of each proposal.

- **The Hardware Evaluation Factors**

When you evaluate the hardware needed by a new business application, you should investigate specific physical and performance characteristics for each computer system or peripheral component to be acquired. Specific questions must be answered concerning many important factors. Ten of these hardware evaluation factors and questions are summarized in Table 2.4.

Table 2.4: Hardware Evaluation Factors

Factor	Description/Example
Performance	What is its speed, capacity, and throughput?
Cost	What is its lease or purchase price? What will be its cost of operation and maintenance?
Reliability	What are the risk of malfunction and its maintenance requirements? What are its error control and diagnostic features?
Compatibility	Is it compatible with existing hardware and software? Is it compatible with hardware and software provided by competing suppliers?
Technology	In what year of its product life cycle is it? Does it use a new untested technology, or does it run the risk of obsolescence?
Ergonomics	Has it been "human factors engineered" with the user in mind? Is it user-friendly, designed to be safe, comfortable, and easy to use?
Connectivity	Can it be easily connected to wide area and local area networks that use different types of network technologies and bandwidth alternatives?
Scalability	Can it handle the processing demands of a wide range of end users, transactions, queries, and other information processing requirements?
Software	Are system and application software available that can best use this hardware?
Support	Are the services required to support and maintain it available?

- **Software Evaluation Factors**

You should evaluate software according to many factors that are similar to those used for hardware evaluation. Thus, the factors of performance, cost, reliability, availability, compatibility, modularity, technology, ergonomics, and support should be used to evaluate proposed software acquisitions. In addition, however, the software evaluation factors summarized in Table 2.5 must also be considered. You should answer the questions they generate in order to evaluate software purchases properly. For example, some software packages are notoriously slow, hard to use, bug-filled, or poorly documented. They are not a good choice, even if offered at attractive prices.

Table 2.5: Software Evaluation Factors

Factor	Description/Example
Quality	Is it bug-free, or does it have many errors in its program code?
Efficiency	Is the software a well-developed system of program code that does not use much CPU time, memory capacity, or disk space?
Flexibility	Can it handle our business processes easily, without major modification?
Security	Does it provide control procedures for errors, malfunctions, and improper use?
Connectivity	Is it Web-enabled so it can easily access the Internet, intranets, and extranets, on its own, or by working with Web browsers or other network software?
Maintenance	Will new features and bug fixes be easily implemented by our own software developers?
Documentation	Is the software well documented? Does it include help screens and helpful software agents?
Hardware	Does existing hardware have the features required to best use this software?
Other Factors	What are its performance, cost, reliability, availability, compatibility, modularity, technology, ergonomics, scalability, and support characteristics? (Use the hardware evaluation factor questions in Table 2.4).

- **IS Services Evaluation Factors**

Most suppliers of hardware and software products and many other firms ***offer*** a variety of IS services to end users and organizations. Examples include assistance in developing a company Web site; installation or conversion of new hardware and software; employee training; and hardware maintenance.

Some of these services are provided without cost by hardware manufacturers and software suppliers.

Other types of IS services needed by a business can be ***outsourced*** to an outside company for a negotiated price. For example, ***systems integrators*** take over complete responsibility for an organization's computer facilities when an organization outsources its computer operations. They may also assume responsibility for developing and implementing large systems development projects that involve many vendors and subcontractors. ***Value-added resellers (VARs)*** specialize in providing industry-specific hardware, software, and services from selected manufacturers. Many other services are available to end users, including systems design, contract programming, and consulting services. Evaluation factors and questions for IS services are summarized in Table 2.6.

Table 2.6: Evaluation Factors for IS Services.

Factor	Description/Example
Performance	What has been their past performance in view of their past promises?
Systems Development	Are Web site and other e-business developers available? What are their quality and cost?
Maintenance	Is equipment maintenance provided? What are its quality and cost?
Conversion	What systems development and installation services will they provide during the conversion period?
Training	Is the necessary training of personnel provided? What are its quality and cost?
Backup	Are similar computer facilities available nearby for emergency backup purposes?
Accessibility	Does the vendor provide local or regional sites that offer sales, systems development, and hardware maintenance services? Is a customer support center at the vendor's Web site available? Is a customer hotline provided?
Business Position	Is the vendor financially strong, with good industry market prospects?
Hardware	Do they provide a wide selection of compatible hardware devices and accessories?
Software	Do they offer a variety of useful e-business software and application packages?

- **Developing**

In this phase/step, the analyst works with programmers to develop any original software that is needed. Programmers have a key role in this phase because they design, code, and remove syntactical errors from computer programs. To ensure quality, a programmer may conduct either a design or a code walkthrough, explaining complex portions of the program to a team of other programmers.

- **Testing**

System testing may involve testing and ***debugging*** software, testing Web site performance, and testing new hardware. An important part of testing is the ***review*** of prototypes of displays, reports, and other output. Prototypes should be reviewed by end users of the proposed systems for possible errors. Of course, testing should not occur only during the system's implementation stage, but throughout the system's development process. For example, you might examine and critique prototypes of input documents, screen displays, and processing procedures during the systems design stage. Immediate end-user testing is one of the benefits of a prototyping process.

Before the information system can be used, it must be tested. It is much less costly to catch problems before the system is signed over to users. Some of the testing is completed by programmers alone, some of it by systems analysts in conjunction with programmers. A series of tests to pinpoint problems is run first with sample data and eventually with actual data from the current system. Often test plans are created early in the SDLC and are refined as the project progresses.

There are three basic types of test as the following***: (1) Unit testing***, or program testing, consists of testing each program separately in the system. ***(2) System testing***, tests the functioning of the information system as a whole. ***(2) Acceptance testing***, provides the final certification that the system is ready to be used in a production setting.

- **Data conversion**

Implementing new information systems for many organizations today frequently involves replacing a previous system and its software and databases. One of the most important implementation activities required when installing new software in such cases is called data conversion. For example, installing new software packages may require converting the data elements in

databases that are affected by a new application into new data formats. Other data conversion activities that are typically required include correcting incorrect data, filtering out unwanted data, consolidating data from several databases, and organizing data into new data subsets, such as databases, data marts, and data warehouses. A good data conversion process is essential because improperly organized and formatted data are frequently reported to be one of the major causes of failures in implementing new systems.

During the design phase, the analysts created a ***data dictionary*** that not only describes the various data elements contained in the new system but also specifies any necessary conversions from the old system. In some cases, only the name of the data element is changed, as in the old system field CUST_ID becoming CLIENT_ID in the new system. In other cases, the actual format of the data is changed, thus requiring some conversion application to be written to filter the old data and put them into the new format. An example of this might be the creation of a new CUSTOMER_ID format to allow for expansion or to make two merged systems compatible with one another. This type of data element conversion requires additional time to occur because each element must be passed through the conversion filter before being written into the new data files.

Yet another issue is the time necessary to transfer the data from the old data files into the files for the new system and the time necessary for the preservation of the integrity of the current system data files during the process. Although it is possible that the new system may have been designed to use the existing data files, this is not normally the case, especially in situations where a new system is replacing a legacy system that is fairly old. The time necessary to transfer the old data can have a material impact on the conversion process and on the strategy that is ultimately selected.

- **System Conversion Strategies**

The analyst needs to plan for a smooth ***conversion*** from the old system to the new one. This process includes converting files from old formats to new ones, or building a database, installing equipment, and bringing the new system into production.

The initial operation of a new business system can be a difficult task. This typically requires a conversion process from the use of a present system to the operation of a new or improved application. Conversion methods can soften the impact of introducing new information technologies into an organization.

Four major forms of system conversion are illustrated in Figure 2.6 . They include (1) Parallel conversion, (2) Phased conversion, (3) Pilot conversion, and (4) Direct conversion.

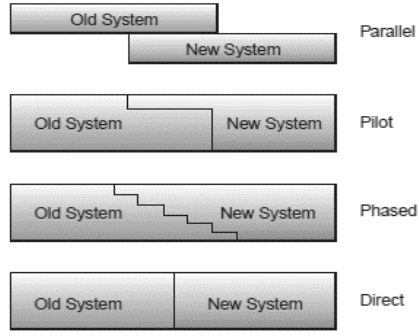


Figure 2.6: The four major forms of conversion to a new system.

1. Direct Conversion

The simplest conversion strategy, and probably the most disruptive to the organization, is the ***direct cutover*** approach. This method, sometimes referred to as the ***slam dunk*** or ***cold-turkey strategy***, is as abrupt as its name implies. Using this approach, the old system is just turned off, and the new system is turned on in its place. Although this method is the least expensive of all available strategies and may be the only viable solution in situations where activating the new system is an emergency or when the two systems cannot coexist under any conditions, it is also the one that poses the greatest risk of failure. Once the new system becomes operational, the end users must cope with any errors or dysfunctions, and depending on the severity of the problem, this approach can have a significant effect on the quality of the work performed. Direct conversion should be considered only in extreme circumstances where no other conversion strategy is viable.

2. Parallel Conversion

At the opposite end of the risk spectrum is the parallel conversion strategy. Here, the old and new systems are run simultaneously until the end users and project coordinators are fully satisfied that the new system is functioning correctly and the old system is no longer necessary. Using this approach, a parallel conversion can be effected with either a ***single cutover***, where a predetermined date for stopping the parallel operation is set, or a ***phased cutover***, where some predetermined method of phasing in each piece of the new system and turning off a similar piece of the old system is employed.

Although clearly having the advantage of ***low risk***, the parallel approach also brings with it the highest cost. To execute a parallel approach properly, the end users must literally perform all daily functions with both systems, thus creating a massive redundancy in

activities and literally double the work. In fact, unless the operational costs of the new system are significantly less than the old system, the cost of parallel operation can be as much as three to four times greater than the old system alone. During a parallel conversion, all outputs from both systems are compared for concurrency and accuracy, until it is determined that the new system is functioning at least as well as the one it is replacing. Parallel conversion may be the best choice in situations where an automated system is replacing a manual one. In certain circumstances where end users cannot cope with the often-confusing redundancy of two systems, the parallel conversion strategy may not be viable. Also, parallel conversion may not be possible if the organization does not have the available computing resources to operate two systems at the same time.

3. Pilot Conversion

In some situations, the new system may be installed in multiple locations, such as a series of bank branches or retail outlets. In other cases, the conversion may be able to be planned from a geographic perspective. When these types of scenarios exist, the possibility of using a ***pilot conversion*** strategy exists. This approach allows for the conversion to the new system, using either a direct or parallel method, at a single location. The advantage to this approach is that a location can be selected that best represents the conditions across the organization but also may be less risky in terms of any loss of time or delays in processing. Once the installation is complete at the pilot site, the process can be evaluated and any changes to the system made to prevent problems encountered at the pilot site from reoccurring at the remaining installations. This approach may also be required if the individual sites or locations have certain unique characteristics or idiosyncrasies making either a direct or parallel approach infeasible.

4. Phased Conversion

A ***phased*** or ***gradual conversion*** strategy attempts to take advantage of the best features of both the direct and parallel approaches, while minimizing the risks involved. This incremental approach to conversion allows for the new system to be brought online as a series of functional components that are

logically ordered to minimize disruption to the end users and the flow of business.

Phased conversion is analogous to the release of multiple versions of an application by a software developer. Each version of the software should correct any known bugs and should allow for 100 percent compatibility with data entered into or processed by the previous version. Although it has the advantage of lower risk, the phased approach takes the most time and, thus, creates the most disruption to the organization over time.

- **Documentation**

Developing good user documentation is an important part of the implementation process. During this phase the analyst works with users to develop effective documentation for software, including procedure manuals, online help, and Web sites featuring Frequently Asked Questions (FAQs), on Read Me files shipped with new software. Because users are involved from the beginning, phase documentation should address the questions they have raised and solved jointly with the analyst. Documentation tells users how to use software and what to do if software problems occur.

Sample data entry display screens, forms, and reports are good examples of documentation. When ***computer-aided systems engineering (CASE)*** methods are used, documentation can be created and changed easily because it is stored and accessible on disk in a ***system repository***. Documentation serves as a method of communication among the people responsible for developing, implementing, and maintaining a computer-based system. Installing and operating a newly designed system or modifying an established application requires a detailed record of that system's design. Documentation is extremely important in diagnosing errors and making changes, especially if the end users or systems analysts who developed a system are no longer with the organization.

- **Training**

Training is a vital implementation activity. This phase/step involves ***training*** users to handle the system. Vendors do some training, but oversight of training is the responsibility of the systems analyst.

IS personnel, such as user consultants, must be sure that end users are trained to operate a new business system or its implementation will fail. Training may involve only activities like data entry, or it may also involve all

aspects of the proper use of a new system. In addition, managers and end users must be educated in how the new technology affects the company's business operations and management. This knowledge should be supplemented by training programs for any new hardware devices, software packages, and their use for specific work activities.

- **Evaluating the System**

Evaluation is included as part of this final phase of the SDLC mostly for the sake of discussion. Actually, evaluation takes place during every phase. A key criterion that must be satisfied is whether the intended users are indeed using the system.

It should be noted that systems work is often cyclical. When an analyst finishes one phase of systems development and proceeds to the next, the discovery of a problem may force the analyst to return to the previous phase and modify the work done there.

2.3.1.5 Systems Maintenance

Maintenance of the system begins in this phase and is carried out routinely throughout the life of the information system. When all is said and done, the single most costly activity occurs after the system implementation is complete: the ***postimplementation maintenance phase***. The primary objectives associated with systems maintenance are to ***correct errors or faults*** in the system, provide changes to effect performance improvement, or adapt the system to changes in the operating or business environment. In a typical organization, more programmers and analysts are assigned to application maintenance activities than to application development. Further, although a new system can take several months or years to design and build and can cost hundreds of thousands or millions of dollars, the resulting system can operate around the clock and last for 5 to 10 years, or longer. One major activity in postimplementation involves making changes to the system after the users have finally had an opportunity to use it. These are called ***change requests***. Such requests can range from fixing a software bug not found during testing to designing an enhancement to an existing process or function.

Managing and implementing change requests is only one aspect of the systems maintenance phase activities. In some ways, once the maintenance phase begins, the life cycle starts over again. New requirements are articulated, analyzed, designed, checked for feasibility, tested, and implemented.

Much of the programmer's routine work consists of maintenance, and businesses spend a great deal of money on maintenance. Some maintenance, such as program updates, can be done automatically via a vendor site on the Web. Many of the systematic procedures the analyst employs throughout the SDLC can help ensure that maintenance is kept to a minimum.

Maintenance is performed for two reasons. The first of these is to ***correct software errors***. No matter how thoroughly the system is tested, bugs or errors creep into computer programs. Bugs in commercial PC software are often documented as "known anomalies," and are corrected when new versions of the software are released or in an interim release. In custom software (also called bespoke software), bugs must be corrected as they are detected.

The other reason for performing system maintenance is to ***enhance the software's capabilities*** in response to changing organizational needs, generally involving one of the following three situations:

1. Users often request additional features after they become familiar with the computer system and its capabilities.
2. The business changes over time.
3. Hardware and software are changing at an accelerated pace.

over time, the total cost of maintenance is likely to exceed that of systems development. At a certain point it becomes more feasible to perform a new systems study, because the cost of continued maintenance is clearly greater than that of creating an entirely new one.

In summary, maintenance is an ongoing process over the life cycle of an information system. After the information system is installed, maintenance usually takes the form of correcting previously undetected program errors. Once these are corrected, the system approaches a steady state, providing dependable service to its users. Maintenance during this period may consist of removing a few previously undetected bugs and updating the system with a few minor enhancements. As time goes on and the business and technology change, however, the maintenance effort increases dramatically.

Although the range and nature of specific maintenance requests vary from system to system, four basic categories of maintenance can be identified: (1) corrective, (2) adaptive, (3) perfective, and (4) preventive.

1. The activities associated with ***corrective maintenance*** are focused on fixing bugs and logic errors not detected during the implementation testing period.

2. ***Adaptive maintenance*** refers to those activities associated with modifying existing functions or adding new functionality to accommodate changes in the business or operating environments.
3. ***Perfective maintenance*** activities involve changes made to an existing system that are intended to improve the performance of a function or interface.
4. The final category of maintenance activities, ***preventive maintenance***, involves those activities intended to reduce the chances of a system failure or extend the capacity of a current system's useful life.

Although often the lowest-priority maintenance activity, preventive maintenance is, nonetheless, a high-value-adding function and is vital to an organization realizing the full value of its investment in the system.

The maintenance activity also includes a ***postimplementation review process*** to ensure that newly implemented systems meet the business objectives established for them. Errors in the development or use of a system must be corrected by the maintenance process. This includes a periodic review or audit of a system to ensure that it is operating properly and meeting its objectives. This audit is in addition to continually monitoring a new system for potential problems or necessary changes.

2.3.2 Using CASE Tools

Analysts who adopt the SDLC approach often benefit from productivity tools, called ***Computer Aided Software Engineering (CASE)*** tools, that have been created explicitly to improve their routine work through the use of automated support. Analysts rely on CASE tools to increase productivity, communicate more effectively with users, and integrate the work that they do on the system from the beginning to the end of the life cycle.

Visible Analyst (VA) and another software product called Microsoft Visio allow users to draw and modify diagrams easily and to do graphical planning, analysis, and design in order to build complex client/server applications and databases Through the use of automated support featuring onscreen output, clients can readily see how data flows and other system concepts are depicted, and they can then request corrections or changes that would have taken too much time with older tools.

Some analysts distinguish between upper and lower CASE tools, as the following:

1) ***Upper CASE tool*** allows the analyst to create and modify the system design. All the information about the project is stored in an encyclopedia called the ***CASE repository***, a large collection of records, elements, diagrams, screens, reports, and other information, Figure 2.7. Analysis reports may be produced using the repository information to show where the design is incomplete or contains errors. Upper CASE tools can also help support the modeling of an organization's functional requirements, assist analysts and users in drawing the boundaries for a given project, and help them visualize how the project meshes with other parts of the organization.

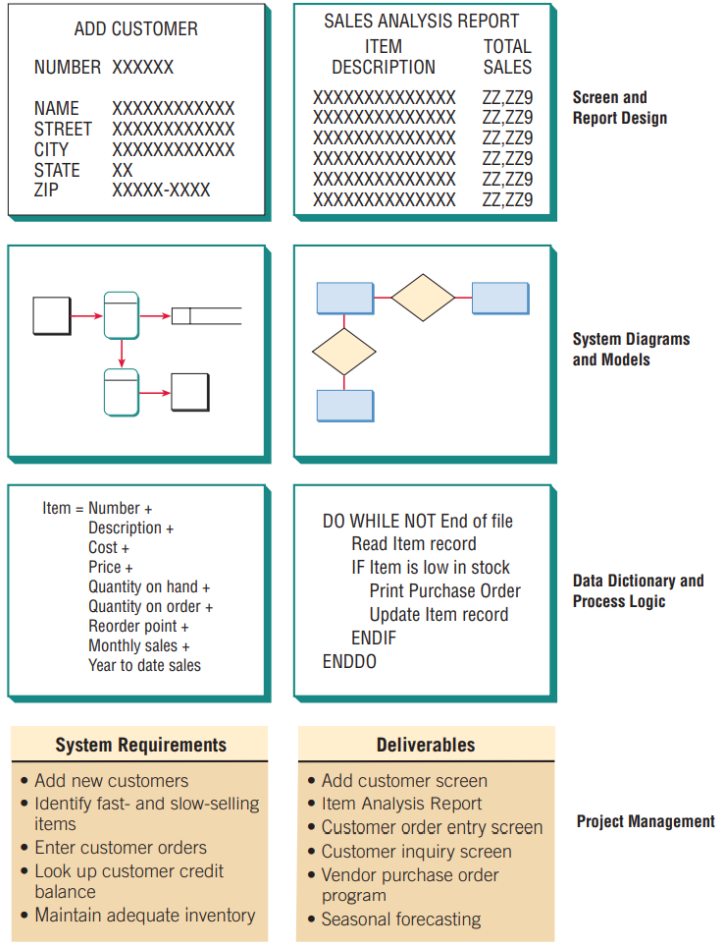


Figure 2.7: The repository concept.

2) ***Lower CASE tools*** are used to generate computer source code, eliminating the need for programming the system. Code generation has several advantages: (1) the system can be produced more quickly than by writing computer programs; (2) the amount of time spent on maintenance decreases with code generation; (3) code can be generated in more than one computer language, so it is easier to migrate systems from one platform to another; (4) code generation provides a cost-effective way of tailoring systems purchased from third-party vendors to the needs of the organization; and (5) generated code is free of computer program errors.

2.3.3 The Agile Approach

Agile development is an object-oriented approach (OOA) to systems development that includes a method of development (including generating information requirements) as well as software tools.

Agile methods are a collection of innovative, user-centered approaches to systems development. You will learn the values and principles, activities, resources, practices, processes, and tools associated with agile methodologies in the upcoming section. Agile methods can be credited with many successful systems development projects and in numerous cases even credited with rescuing companies from a failing system that was designed using a structured methodology.

2.3.3.1 Developmental Process for an Agile Project

Two words that characterize a project done with an agile approach are interactive and incremental, as shown in Figure 2.8.

By examining Figure 2.8 we can see that there are five distinct stages: exploration, planning, iterations to the first release, productionizing, and maintenance. Notice that the three red arrows that loop back into the "Iterations" box symbolize incremental changes created through repeated testing and feedback that eventually lead to a stable but evolving system. Also note that there are multiple looping arrows that feed back into the productionizing phase. These symbolize that the pace of iterations is increased after a product is released. The red arrow is shown leaving the maintenance stage and returning to the planning stage, so that there is a continuous feedback loop involving customers and the development team as they agree to alter the evolving system.

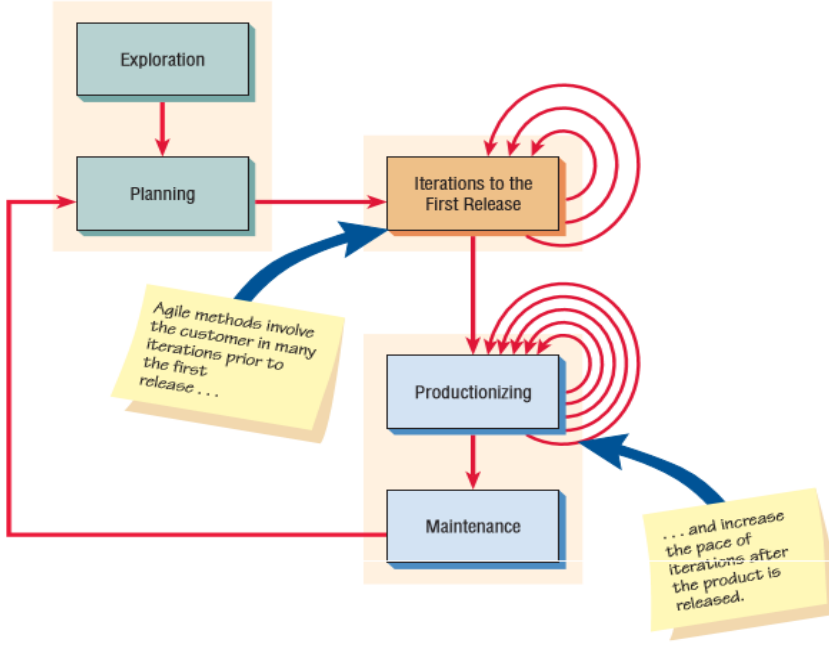


Figure 2.8: The Agile Modeling Development Process.

- **Exploration.**

During exploration, you will explore your environment, asserting your conviction that the problem can and should be approached with agile development, assemble the team, and assess team member skills. This stage will take anywhere from a few weeks (if you already know your team members and technology) to a few months (if everything is new). You also will be actively examining potential technologies needed to build the new system. During this stage you should practice estimating the time needed for a variety of tasks. In exploration, customers also are experimenting with writing user stories. The point is to get the customer to refine a story enough so that you can competently estimate the amount of time it will take to build the solution into the system you are planning. This stage is all about adopting a playful and curious attitude toward the work environment, its problems, technologies, and people.

- **Planning.**

In contrast to the first stage, planning may only take a few days to accomplish. In this stage you and your customers agree on a date anywhere

from two months to half a year from the current date to deliver solutions to their most pressing business problems (you will be addressing the smallest, most valuable set of stories). If your exploration activities were sufficient, this stage should be very short.

The entire agile planning process has been characterized using the idea of a ***planning game***. The planning game spells out rules that can help formulate the agile development team's relationship with their business customers. Although the rules form an idea of how you want each party to act during development, they are not meant as a replacement for a relationship. They are a basis for building and maintaining a relationship.

So, we use the metaphor of a game. To that end we talk in terms of the goal of the game, the strategy to pursue, the pieces to move, and the players involved. The goal of the game is to maximize the value of the system produced by the agile team. In order to figure the value, you have to deduct costs of development, and the time, expense, and uncertainty taken on so that the development project could go forward.

The strategy pursued by the agile development team is always one of limiting uncertainty (downplaying risk). To do that they design the simplest solution possible, put the system into production as soon as possible, get feedback from the business customer about what's working, and adapt their design from there. Story cards become the pieces in the planning game that briefly describe the task, provide notes, and provide an area for task tracking.

There are two main players in the planning game: the ***development team*** and the ***business customer***. Deciding which business group in particular will be the business customer is not always easy, because the agile process is an unusually demanding role for the customer to play. Customers decide what the development team should tackle first. Their decisions will set priorities and check functionalities throughout the process.

- **Iterations to the First Release.**

Typically these are iterations (cycles of testing, feedback, and change) of about three weeks in duration. You will be pushing yourself to sketch out the entire architecture of the system, even though it is just in outline or skeletal form. One goal is to run customer-written functional tests at the end of each iteration. During the iterations stage you should also question whether the schedule needs to be altered or whether you are tackling too many stories. Make small services out of each successful iteration, involving customers as

well as developers. Always celebrate your progress, even if it is small, because this is part of the culture of motivating everyone to work extremely hard on the project.

- **Productionizing.**

In this phase the feedback cycle speeds up so that rather than receiving feedback for an iteration every three weeks, software revisions are being turned around in one week. You may institute daily briefings so everyone knows what everyone else is doing. The product is released in this phase, but may be improved by adding other features. Getting a system into production is an exciting event. Make time to celebrate with your teammates and mark the occasion. One of the watchwords of the agile approach, with which we heartily agree, is that it is supposed to be fun to develop systems!

- **Maintenance.**

Once the system has been released, it needs to be kept running smoothly. New features may be added, riskier customer suggestions may be considered, and team members may be rotated on or off the team.

2.3.4 Object–Oriented Systems

An object-oriented system is composed of objects. An ***object*** can be anything a programmer wants to manage or manipulate—cars, people, animals, savings accounts, food products, business units, organizations, customers—literally anything. Once an object is defined by a programmer, its characteristics can be used to allow one object to interact with another object or pass information to another object. The behavior of an object-oriented system entails collaboration between these objects, and the state of the system is the combined state of all the object in it.

Collaboration between objects requires them to send messages or information to one another. The exact semantics of message sending between objects varies, depending on the kind of system being modeled. In some systems, "sending a message" is the same as "invoking a method." In others, "sending a message" might involve sending data using a pre-prescribed media. The three areas of interest to us in an object-oriented system are object-oriented programming, object-oriented analysis, and object-oriented design.

2.3.4.1 Object–Oriented Programming (OOP)

Object-oriented programming (OOP) is the programming paradigm that uses "objects" to design applications and computer programs. It differs from traditional procedural programming by examining objects that are part of a system. Each ***object*** is a computer representation of some actual thing or event. Objects may be customers, items, orders, and so on. Objects are represented by and grouped into classes that are optimal for reuse and maintainability. A ***class*** defines the set of shared attributes and behaviors found in each object in the class.

OOP employs several techniques from previously established paradigms, including:

- ***Inheritance***. The ability of one object to inherit the characteristics of a higherorder object. For example, all cars have wheels; therefore, an object defined as a sports car and as a special type of the object cars must also have wheels.
- ***Modularity***. The extent to which a program is designed as a series of interlinked yet stand-alone modules.
- ***Polymorphism***. The ability of an object to behave differently depending on the conditions in which its behavior is invoked. For example, two objects that inherit the behavior speak from an object class animal might be a dog object and a cat object. Both have a behavior defined as speak. When the dog object is commanded to speak, it will bark, whereas when the cat object is commanded to speak, it will meow.
- ***Encapsulation***. Concealing all of the characteristics associated with a particular object inside the object itself. This paradigm allows objects to inherit characteristics simply by defining a subobject. For example, the object airplane contains all of the characteristics of an airplane: wings, tail, rudder, pilot, speed, altitude, and so forth.

2.3.4.2 Object–Oriented Analysis (OOA)

It aims to model the problem domain, that is, the problem we want to solve, by developing an object-oreinted (OO) system. The source of the analysis is a set of written requirements statements and/or diagrams that illustrate the statements.

Similar to the SDLC-developed model, an object-oriented analysis model does not take into account implementation constraints, such as concurrency, distribution, persistence, or inheritance, nor how the system will be built.

Because object-oriented systems are modular, the model of the system can be divided into multiple domains, each of which are separately analyzed and represent separate business, technological, or conceptual areas of interest. The result of object-oriented analysis is a description of what is to be built, using concepts and relationships between concepts, often expressed as a conceptual model. Any other documentation needed to describe what is to be built is also included in the results of the analysis.

2.3.4.3 Object–Oriented Design (OOD)

It describes the activity when designers look for logical solutions to solve a problem using objects. Object-oriented design takes the conceptual model that results from the object-oriented analysis and adds implementation constraints imposed by the environment, the programming language, and the chosen tools, as well as architectural assumptions chosen as the basis of the design.

The concepts in the conceptual model are mapped to concrete classes, abstract interfaces, and roles that the objects take in various situations. The interfaces and their implementations for stable concepts can be made available as reusable services. Concepts identified as unstable in OOA will form the basis for policy classes that make decisions and implement environment or situation-specific logics or algorithms. The result of OOD is a detailed description of how the system can be built, using objects.

Thus, the object-oriented world bears many simularities to the more conventional SDLC approach. This approach simply takes a different view of the programming domain and thus approaches the problem-solving activities inherent in system development from a different direction

2.3.4.4 Object–Oriented Systems Analysis and Design

Object-oriented (O-O) analysis and design is an approach that is intended to facilitate the development of systems that must change rapidly in response to dynamic business environments.

Object-oriented techniques are thought to work well in situations in which complicated information systems are undergoing continuous maintenance, adaptation, and redesign. Object-oriented approaches use the industry standard for modeling object-oriented systems, called the ***Unified Modeling Language (UML)***, to break down a system into a use case model.

The phases in UML are similar to those in the SDLC. Since those two methods share rigid and exacting modeling, they happen in a slower, more deliberate pace than the phases of agile modeling. The analyst goes through problem and identification phases, an analysis phase, and a design phase as shown in Figure 2.10.

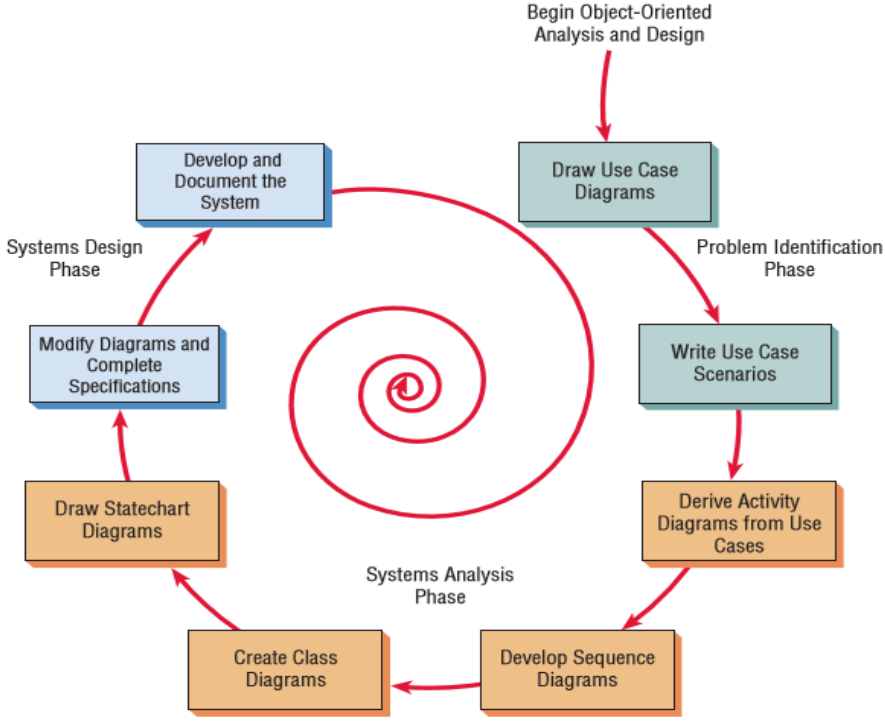


Figure 2.9: The steps in the UML development process.

1- ***Define the use case model.*** In this phase the analyst identifies the actors and the major events initiated by the actors. Often the analyst will start by drawing a diagram with stick figures representing the actors and arrows showing how the actors relate. This is called a use case diagram and it represents the standard flow of events in the system. Then an analyst typically writes up a use case scenario, which describes in words the steps that are normally performed.

2- ***The systems analysis phase,*** begin drawing UML diagrams. In the second phase, the analyst will draw Activity Diagrams, which illustrate all the major activities in the use case. In addition, the analyst will create one or more sequence diagrams for each use case, which show

the sequence of activities and their timing. This is an opportunity to go back and review the use cases, rethink them, and modify them if necessary.

3- Continuing in the analysis phase, ***develop class diagrams*.** The nouns in the use cases are objects that can potentially be grouped into classes. e.g, every automobile is an object that shares characteristics with other automobiles. Together they make up a class.
4- Still in the analysis phase, ***draw statechart diagrams*.** The class diagrams are used to draw statechart diagrams, which help in understanding complex processes that cannot be fully derived by the sequence diagrams. The statechart diagrams are extremely useful in modifying class diagrams, so the iterative process of UML modeling continues.
5- Begin ***systems design*** by modifying the UML diagrams. Then complete the specifications. Systems design means modifying the existing system and that implies modifying the diagrams drawn in the previous phase. These diagrams can be used to derive classes, their attributes, and methods (***methods are simply operations***). The analyst will need to write class specifications for each class including the attributes, methods, and their descriptions. They will also develop methods specifications that detail the input and output requirements for the method, along with a detailed description of the internal processing of the method.
6- ***Develop and document the system*.** Documentation is critical. The more complete the information you provide the development team through documentation and UML diagrams, the faster the development and the more solid the final production system.

Object-oriented methodologies often focus on small, quick iterations of development, sometimes called the ***spiral model***. Analysis is performed on a small part of the system, usually starting with a high-priority item or perhaps one that has the greatest risk. This is followed by design and implementation. The cycle is repeated with analysis of the next part, design, and some implementation, and it is repeated until the project is completed. Reworking diagrams and the components themselves is normal. UML is a powerful modeling tool that can greatly improve the quality of your systems analysis and design and the final product.

2.3.5 Prototyping

Prototyping is the rapid development and testing of working models, or prototypes, of new applications in an interactive, iterative process that can be used by both IS specialists and business professionals. Prototyping, as a development tool, makes the development process faster and easier, especially for projects where end-user requirements are hard to define. Prototyping has also opened up the application development process to end users because it simplifies and accelerates systems design. Thus, prototyping has enlarged the role of the business stakeholders affected by a proposed system and helps make possible a quicker and more responsive development process called ***agile systems development (ASD)***.

As the systems analyst presenting a prototype of the information system, you are keenly interested in the reactions of users and management to the prototype. You want to know in detail how they react to working with the prototype and how good the fit is between their needs and the prototyped features of the system. Reactions are gathered through observation, interviews, and feedback sheets (possibly questionnaires) designed to elicit each person's opinion about the prototype as he or she interacts with it. Information gathered in the prototyping phase allows the analyst to set priorities and redirect plans inexpensively, with a minimum of disruption. Because of this feature, prototyping and planning go hand-in-hand.

2.3.5.1 The Prototyping Process

Prototyping can be used for both large and small applications. Typically, large business systems still require using a traditional systems development approach, but parts of such systems can frequently be prototyped. A prototype of a business application needed by an end user is developed quickly using a variety of application development software tools. The prototype system is then repeatedly refined until it is acceptable.

As Figure 2.11 illustrates, prototyping is an iterative, interactive process. End users with sufficient experience with application development tools can do prototyping themselves. Alternatively, you could work with an IS specialist to develop a prototype system in a series of interactive sessions. For example, you could develop, test, and refine prototypes of management reports, data entry screens, or output displays.

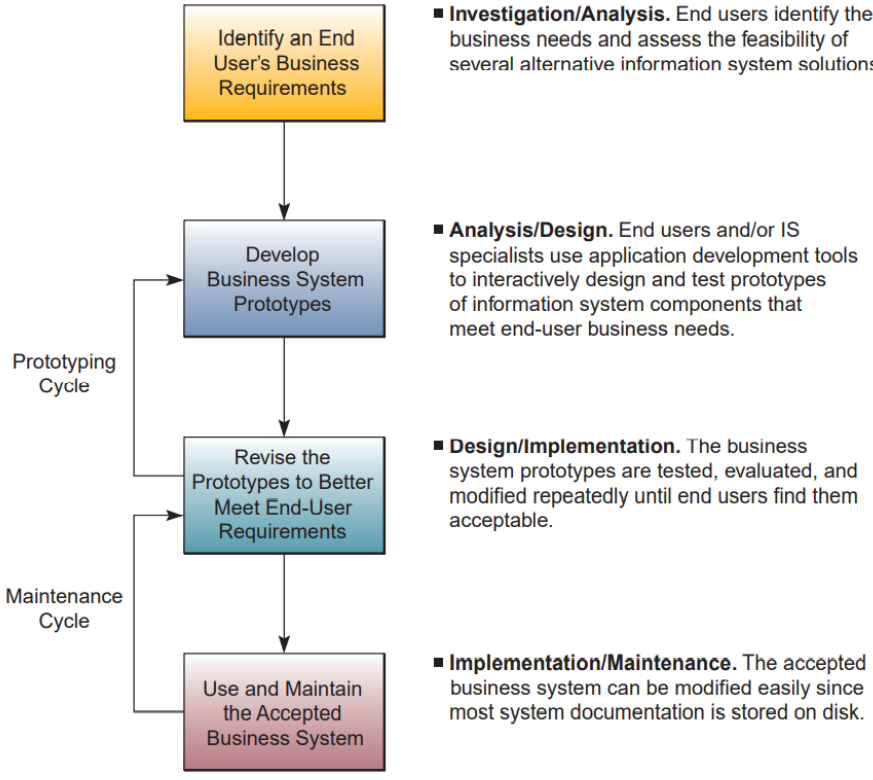


Figure 2.10: Application development using prototyping.

Usually, a prototype is modified several times before end users find it acceptable. Program modules are then generated by application development software using conventional programming languages. The final version of the application system is then turned over to its end users for operational use. While prototyping is a useful method of allowing an end user to develop small software applications, its real power is as a development tool, within a life cycle project, to help analysts and users finalize the various interfaces and functions of a large business system.

2.3.5.2 Guidelines for Developing a Prototype

The guidelines suggest ways of proceeding with the prototype that are necessarily interrelated. Once the decision to prototype has been made, four main guidelines must be observed when integrating prototyping into the

requirements determination phase of the SDLC. Each guideline is explained in the following subsections.

- **Work in Manageable Modules**

When prototyping some of the features of a system into a workable model, it is imperative that the analyst work in manageable modules. One distinct advantage of prototyping is that it is not necessary or desirable to build an entire working system for prototype purposes.

A manageable module is one that allows users to interact with its key features but can be built separately from other system modules. Module features that are deemed less important are purposely left out of the initial prototype. As you will see later in this chapter, this is very similar to the agile approach that emphasizes small releases.

- **Build the Prototype Rapidly**

Speed is essential to the successful prototyping of an information system. Recall that one complaint voiced against following the traditional SDLC is that the interval between requirements determination and delivery of a complete system is far too long to address evolving user needs effectively.

Analysts can use prototyping to shorten this gap by using traditional information-gathering techniques to pinpoint salient information requirements, and then quickly make decisions that bring forth a working model. In effect the user sees and uses the system very early in the SDLC instead of waiting for a finished system to gain hands-on experience.

Putting together an operational prototype both rapidly and early in the SDLC allows the analyst to gain valuable insight into how the remainder of the project should go. By showing users very early in the process how parts of the system actually perform, rapid prototyping guards against overcommitting resources to a project that may eventually become unworkable.

- **Modify the Prototype in Successive Iterations**

A third guideline for developing the prototype is that its construction must support modifications. Making the prototype modifiable means creating it in modules that are not highly interdependent. If this guideline is observed, less resistance is encountered when modifications in the prototype are necessary.

The prototype is generally modified several times, going through several iterations. Changes in the prototype should move the system closer to what

users say is important. Each modification necessitates another evaluation by users.

The prototype is not a finished system. Entering the prototyping phase with the idea that the prototype will require modification is a helpful attitude that demonstrates to users how necessary their feedback is if the system is to improve.

- **Stress the User Interface**

The user's interface with the prototype (and eventually the system) is very important. Because what you are really trying to achieve with the prototype is to get users to further articulate their information requirements, they must be able to interact easily with the system's prototype. They should be able to see how the prototype will enable them to accomplish their tasks. For many users the interface is the system. It should not be a stumbling block.

Although many aspects of the system will remain undeveloped in the prototype, the user interface must be well developed enough to enable users to pick up the system quickly and not be put off. Online, interactive systems using GUI interfaces are ideally suited to prototypes.

2.3.5.3 Kinds of Prototypes

The word prototypeis used in many different ways. Rather than attempting to synthesize all these uses into one definition or trying to mandate one correct approach to the somewhat controversial topic of prototyping, we illustrate how each of several conceptions of prototyping may be usefully applied in a particular situation, as shown in Figure 2.12.

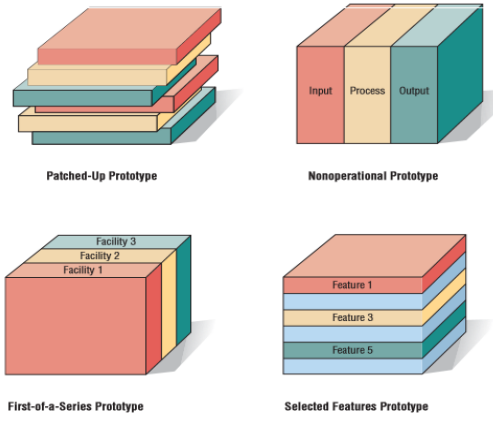


Figure 2.11: kinds of prototypes.

- **Patched–Up Prototype**

The first kind of prototyping has to do with constructing a system that works but is patched up or patched together. In engineering this approach is referred to as ***breadboarding***: creating a patched-together, working model of an (otherwise microscopic) integrated circuit.

An example in information systems is a working model that has all the necessary features but is inefficient. In this instance of prototyping, users can interact with the system, getting accustomed to the interface and types of output available. The retrieval and storage of information may be inefficient, however, because programs were written rapidly with the objective of being workable rather than efficient.

- **Nonoperational Prototype**

The second conception of a prototype is that of a nonworking scale model that is set up to test certain aspects of the design. An example of this approach is a full-scale model of an automobile that is used in wind tunnel tests. The size and shape of the auto are precise, but the car is not operational. In this case only features of the automobile essential to wind tunnel testing are included.

Anonworking scale model of an information system might be produced when the coding required by the applications is too extensive to prototype but when a useful idea of the system can be gained through the prototyping of the input and output only. In this instance, processing, because of undue cost and time, would not be prototyped. Users could still make decisions on the utility of the system, based on their use of prototyped input and output.

- **First–of–a–Series Prototype**

A third conception of prototyping involves creating a first fullscale model of a system, often called a pilot. An example is prototyping the first airplane of a series, then seeing if it flies before building a second. The prototype is completely operational and is a realization of what the designer hopes will be a series of airplanes with identical features.

This type of prototyping is useful when many installations of the same information system are planned. The full-scale working model allows users to experience realistic interaction with the new system, but it minimizes the cost of overcoming any problems that it presents. For example, when a retail grocery chain intends to use electronic data interchange (EDI) to check in

suppliers'shipments in a number of outlets, a full-scale model might be installed in one store so users could work through any problems before the system is implemented in all the others.

- **Selected Features Prototype**

A fourth conception of prototyping concerns building an operational model that includes some, but not all, of the features that the final system will have. An analogy would be a new retail shopping mall that opens before the construction of all shops is complete.

When prototyping information systems in this way, some, but not all, essential features are included. For example, users may view a system menu on a screen that lists six features: add a record, update a record, delete a record, search a record for a key word, list a record, or scan a record. In the prototyped system, however, only three of the six may be available for use, so that the user may add a record (feature 1), delete a record (feature 3), and list a record (feature 5). User feedback can help analysts understand what is working and what isn't. It can also help with suggestions on what features to add next.

When this kind of prototyping is done, the system is accomplished in modules so that if the features that are prototyped are evaluated by users as successful, they can be incorporated into the larger, final system without undertaking immense work in interfacing. Prototypes done in this manner are part of the actual system. They are not just a mock-up as in nonoperational prototyping considered previously. Unless otherwise mentioned, all further references to prototyping in this chapter refer to the selected-features prototype.

- **Advantages of Prototyping**

Prototyping is not necessary or appropriate in every systems project, as we have seen. The advantages, however, should also be given consideration when deciding whether to prototype. The three major advantages of prototyping are:

1) the potential for changing the system early in its development,
2) the opportunity to stop development on a system that is not working,
3) the possibility of developing a system that more closely addresses users'needs and expectations.

Successful prototyping depends on early and frequent user feedback, which analysts can use to modify the system and make it more responsive to actual needs. As with any systems effort, early changes are less expensive than

changes made late in the project's development. In the later part of the chapter, you will see how the agile approach to development uses an extreme form of prototyping that requires an on-site customer to provide feedback during all iterations.

- **Disadvantages of Prototyping**

As with any information-gathering technique, there are several disadvantages to prototyping:

1) It can be quite difficult to manage prototyping as a project in the larger systems effort.
2) Users and analysts may adopt a prototype as a completed system when it is in fact inadequate and was never intended to serve as a finished system.
3) Analysts need to work to ensure that communication with users is clear regarding the timetable for interacting with and improving the prototype.

The analyst needs to weigh these disadvantages against the known advantages when deciding whether to prototype, when to prototype, and how much of the system to prototype.

2.3.6 RAID Application Development

Rapid application development (RAD) is an object-oriented approach to systems development that includes a method of development as well as software tools. It makes sense to discuss RAD and prototyping in the same chapter, because they are conceptually very close. Both have as their goal the shortening of time typically needed in a traditional SDLC between the design and implementation of the information system. Ultimately, both RAD and prototyping are trying to meet rapidly changing business requirements more closely. Once you have learned the concepts of prototyping, it is much easier to grasp the essentials of RAD, which can be thought of as a specific implementation of prototyping.

Some developers are looking at RAD as a helpful approach in new ecommerce, Web-based environments in which so-called first-mover status of a business might be important. In other words, to deliver an application to the Web before their competitors, businesses may want their development team to experiment with RAD.

2.3.6.1 Phases of RAD

There are three broad phases to RAD that engage both users and analysts in assessment, design, and implementation. RAD involves users in each part of the development effort, with intense participation in the business part of the design. Figure 2.13 depicts these three phases.

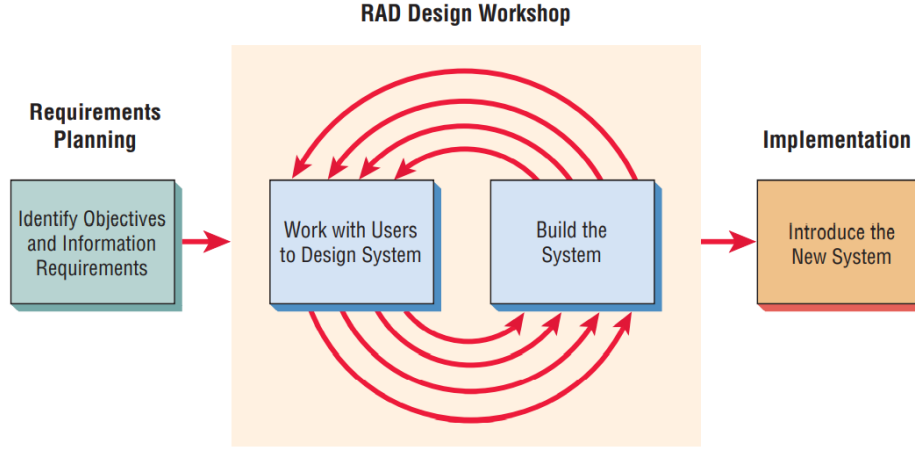


Figure 2.12: The RAD design workshop.

- **Requirements Planning Phase**

In the requirements planning phase, users and analysts meet to identify objectives of the application or system and to identify information requirements arising from those objectives. This phase requires intense involvement from both groups; it is not just signing off on a proposal or document. In addition, it may involve users from different levels of the organization. In the requirements planning phase, when information requirements are still being addressed, you may be working with the CIO (if it is a large organization) as well as with strategic planners, especially if you are working with an ecommerce application that is meant to further the strategic aims of the organization. The orientation in this phase is toward solving business problems. Although information technology and systems may even drive some of the solutions proposed, the focus will always remain on reaching business goals.

- **RAD Design Workshop**

The RAD design workshop phase is a design-and-refine phase that can best be characterized as a workshop. When you imagine a workshop, you know that participation is intense, not passive, and that it is typically hands on. Usually participants are seated at round tables or in a U-shaped configuration of chairs with attached desks where each person can see the other and where there is space to work on a notebook computer. If you are fortunate enough to have a group decision support systems (GDSS) room available at the company or through a local university, use it to conduct at least part of your RAD design workshop. During the RAD design workshop, users respond to actual working prototypes and analysts refine designed modules (using some of the software tools mentioned later) based on user responses. The workshop format is very exciting and stimulating, and if experienced users and analysts are present, there is no question that this creative endeavor can propel development forward at an accelerated rate.

- **Implementation Phase**

In the previous figure, you can see that analysts are working with users intensely during the workshop to design the business or nontechnical aspects of the system. As soon as these aspects are agreed on and the systems are built and refined, the new systems or part of systems are tested and then introduced to the organization. Because RAD can be used to create new ecommerce applications for which there is no old system, there is often no need to (and no real way to) run the old and new systems in parallel before implementation. By this time, the RAD design workshop will have generated excitement, user ownership, and acceptance of the new application. Typically, change brought about in this manner is far less wrenching than when a system is delivered with little or no user participation.

- **When to Use RAD.**

As an analyst, you want to learn as many approaches and tools as possible to facilitate getting your work done in the most appropriate way. Certain applications and systems work will call forth certain methodologies. Consider using RAD when:

1. Your team includes programmers and analysts who are experienced with it; and

2. There are pressing business reasons for speeding up a portion of an application development; or
3. When you are working with a novel ecommerce application and your development team believes that the business can sufficiently benefit over their competitors from being an innovator if this application is among the first to appear on the Web; or
4. When users are sophisticated and highly engaged with the organizational goals of the company.

- **Disadvantages oF RAD.**

The difficulties with RAD, as with other types of prototyping, arise because systems analysts try to hurry the project too much. Suppose two carpenters are hired to build two storage sheds for two neighbors. The first carpenter follows the SDLC philosophy, whereas the second follows the RAD philosophy.

The first carpenter is systematic, inventorying every tool, lawn mower, and piece of patio furniture to determine the correct size for the shed, designing a blueprint of the shed, and writing specifications for every piece of lumber and hardware. The carpenter builds the shed with little waste and has precise documentation about how the shed was built if anyone wants to build another just like it, repair it, or paint it using the same color.

The second carpenter jumps right into the project by estimating the size of the shed, getting a truckload of lumber and hardware, building a frame and discussing it with the owner of the property as modifications are made when certain materials are not available, and making a trip to return the lumber not used. The shed gets built faster, but if a blueprint is not drawn, the documentation never exists.

2.4 Choosing Systems Development Method To Use

The three approaches have all of these activities in, (1) the analyst needs to understand the organization first, (2) the analyst or project team needs to budget their time and resources and develop a project proposal, Next (3) they need to interview organizational members and gather detailed data by using questionnaires and sample data from existing reports and observe how businessis currently transacted.

Even the methods themselves have similarities. The SDLC and object-oriented approaches both require extensive planning and diagramming. The

agile approach and the object-oriented approach both allow subsystems to be built one at a time until the entire system is complete. The agile and SDLC approaches are both concerned a bout the way data logically moves through the system. The following Table 2.7 provides a set of guidelines to help you choose which method to use when developing your next system.

Table 2.7: Choosing Systems Approach Guidline

Choose	When
The Systems Development Life Cycle (SDLC) Approach	• Systems have been developed and documented using SDLC. • It is important to document each step of the way. • Communication of how new systems work is important. • Upper-level management feels more comfortable or safe using SDLC. • There are adequate resources and time to complete the full SDLC.
Agile Methodologies	• There is a project champion of agile methods in the organization. • A rescue takes place (the system failed and there is no time to figure out what went wrong). • Executives and analysts agree with the principles of agile methodologies.
Object-Oriented Methodologies	• The problems modeled lend themselves to classes. • An organization supports the UML learning. • Systems can be added gradually, one subsystem at a time. • Reuse of previously written software is a possibility. • It is acceptable to tackle the difficult problems first.

2.4.1 Comparing Prototyping to the SDLC

Some analysts argue that prototyping should be considered as an alternative to the SDLC. Complaints about going through the SDLC process center around two interrelated concerns.

- The first concern is the extended time required to go through the development life cycle. As the investment of analyst time increases, the cost of the delivered system rises proportionately.
- The second concern about using the SDLC is that user requirements change over time. During the long interval between the time that user requirements are analyzed and the time that the finished system is delivered, user requirements are evolving. Thus, because of the

extended development cycle, the resulting system may be criticized for inadequately addressing current user information requirements.

A corollary of the problem of keeping up with user information requirements is the suggestion that users cannot really know what they do or do not want until they see something tangible. In the traditional SDLC, it often is too late to change an unwanted system once it is delivered.

To overcome these problems, some analysts propose that prototyping be used as an alternative to the SDLC. When prototyping is used in this way, the analyst effectively shortens the time between ascertainment of human information requirements and delivery of a workable system. In addition, using prototyping instead of the traditional SDLC might overcome some of the problems of accurately identifying user information requirements.

Drawbacks to supplanting the SDLC with prototyping include prematurely shaping a system before the problem or opportunity being addressed is thoroughly understood. Also, using prototyping as an alternative may result in producing a system that is accepted by specific groups of users but that is inadequate for overall system needs.

The approach we advocate here is to use prototyping as a part of the traditional SDLC. In this view prototyping is considered as an additional, specialized method for ascertaining users' information requirements as they interact with prototypes and provide feedback for the analyst.

2.4.2 Comparing RAD to the SDLC

The ultimate purpose of RAD is to shorten the SDLC and in this way respond more rapidly to dynamic information requirements of organizations. The SDLC takes a more methodical, systematic approach that ensures completeness and accuracy and has as its intention the creation of systems that are well integrated into standard business procedures and culture.

The RAD design workshop phase is a departure from the standard SDLC design phases, because RAD software tools are used to generate screens and to exhibit the overall flow of the running of the application. Thus, when users approve this design, they are signing off on a visual model representation, not just a conceptual design represented on paper, as is traditionally the case. The implementation phase of RAD is in many ways less stressful than others, because the users have helped to design the business aspects of the system and are well aware of what changes will take place. There are few surprises, and the change is something that is welcomed. Often when using the SDLC, there

is a lengthy time during development and design when analysts are separated from users. During this period, requirements can change and users can be caught off guard if the final product is different than anticipated over many months.

2.5 Open Source Software

An alternative to traditional software development in which proprietary code is hidden from the users is called ***Open Source Software (OSS)***. With OSS, the code, or computer instructions, can be studied, shared, and modified by many users and programmers. Rules of this community include the idea that any program modifications must be shared with all the people on the project.

Development of OSS has also been characterized as a philosophy rather than simply as the process of creating new software. Often those involved in OSS communities view it as a way to help societies change. Widely known open source projects include Apache for developing a Web server, the browser called Mozilla Firefox, and Linux, which is a Unix-like open source operating system.

However, it would be an oversimplification to think of OSS as a monolithic movement, and it does little to reveal what type of users or user analysts are developing OSS projects and on what basis. To help us understand the open source movement, researchers have recently categorized open source communities into four community types—ad hoc, standardized, organized, and commercial—along six different dimensions—general structure, environment, goals, methods, user community, and licensing. Some researchers argue that OSS is at a crossroads and that the commercial and community OSS groups need to understand where they converge and where the potential for conflict exists.

Open source development is useful for many applications running on diverse technologies, including handheld devices and communication equipment. Its use may encourage progress in creating standards for devices to communicate more easily. Widespread use of OSS may alleviate some of the severe shortages of programmers by placing programming tools in the hands of students in developing countries sooner than if they were limited to using proprietary packages, and it may lead to solving large problems through intense and extensive collaboration.

CHAPTER 3
PROJECT MANAGEMENT

3.1 Process Specification and Structured Decisions.

3.1.1 Structured English

When the process logic involves formulas or iteration, or when structured decisions are not complex, an appropriate technique for analyzing the decision process is the use of structured English. As the name implies, structured English is based on

- Structured logic, or instructions organized into nested and grouped procedures, and
- Simple English statements such as add, multiply, and move.

A word problem can be transformed into structured English by putting the decision rules into their proper sequence and using the convention of IF-THEN-ELSE statements throughout.

3.1.2 Writing Structured English

To write structured English, you may want to use the following conventions:

1. Express all logic in terms of one of these four types: sequential structures, decision structures, case structures, or iterations.
2. Use and capitalize accepted keywords such as IF, THEN, ELSE, DO, DO WHILE, DO UNTIL, and PERFORM.
3. Indent blocks of statements to show their hierarchy (nesting) clearly.
4. When words or phrases have been defined in a data dictionary, underline those words or phrases to signify that they have a specialized, reserved meaning.
5. Be careful when using “and” and “or,” and avoid confusion when distinguishing between “greater than” and “greater than or equal to” and like relationships. Clarify the logical statements now rather than waiting until the program coding stage.

3.1.3 Decision Tables

A decision table is a table of rows and columns, separated into four quadrants, as shown in Figure 3.1. The upper left quadrant contains the condition(s); the upper right quadrant contains the condition alternatives. The lower half of the table contains the actions to be taken on the left and the rules

for executing the actions on the right. When a decision table is used to determine which action needs to be taken, the logic moves clockwise beginning from the upper left.

Conditions and Actions	Rules
Conditions	Condition Alternatives
Actions	Action Entries

Figure 3.1: The a decision table standard format.

3.1.3.1 Developing Decision Tables

To build decision tables, the analyst needs to determine the maximum size of the table; eliminate any impossible situations, inconsistencies, or redundancies; and simplify the table as much as possible.

	Rules			
Conditions and Actions	**1**	**2**	**3**	**4**
Under $50	Y	Y	N	N
Pays by check with two forms of ID	Y	N	Y	N
Uses credit card	N	Y	N	Y
Complete the sale after verifying signature.	X			
Complete the sale. No signature needed.		X		
Call supervisor for approval.			X	
Communicate electronically with bank for credit card authorization.				X

Figure 3.2: Using a decision table for illustrating a store's policy of customer checkout with four sets of rules and four possible actions.

Example

Build the decision table which is used to calculate the commission based on (1) the volume of sale, (2) percent repaid, and (3) salary. The calculation is according to te following situations:

1) Commission will be 16% If the volume of sale is greater than 5000$, percent repaid is greater tnan 50%, and the salary of the employee who sold the product is less than 60$.

2) The commission will be 14% if the situation is as same as case 1 but the employee's salary is between 60$ and 120$.
3) The commission will be 12% if the situation is as same as case 1 but the employee's salary is grater than 120$.
4) The commission will be 8% if the situation is as same as case 1 but the percent repaid is less than 50%.
5) The commission will be 7% if the situation is as same as case 2 but the percent repaid is less than 50%.
6) The commission will be 6% if the situation is as same as case 3 but the percent repaid is less than 50%.
7) Commission will be 12% If the volume of sale is less than 5000$, percent repaid is greater tnan 50%, and the salary of the employee who sold the product is less than 60$.
8) The commission will be 11% if the situation is as same as case 7 but the employee's salary is between 60$ and 120$.
9) The commission will be 6% if the situation is as same as case 7 but the percent repaid is less than 50%.
10) The commission will be 8% if the situation is as same as case 8 but the employee's salary is greater than 120$.
11) The commission will be 5% if the situation is as same as case 8 but the percent repaid is less than 50%.
12) The commission will be 4% if the situation is as same as case 7 but the employee's salary is greater then 120$

Answer

Volume of sale>5000$	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y
Percent repaid>50%	N	N	N	Y	Y	Y	N	N	N	Y	Y	Y
Employee's salary	<60	60-120	>60	<60	60-120	>60	<60	60-120	>60	<60	60-120	>60
Case (NO.)	9	11	12	7	8	10	4	5	6	1	2	3
Comission	6%	5%	4%	12%	11%	8%	8%	7%	6%	16%	14%	12%

3.1.4 Decision Trees

Decision trees are used when complex branching occurs in a structured decision process. Trees are also useful when it is essential to keep a string of decisions in a particular sequence. Unlike the decision tree used in

management science, the analyst's tree does not contain probabilities and outcomes. In systems analysis, trees are used mainly for identifying and organizing conditions and actions in a completely structured decision process

Drawing Decision Trees

It is useful to distinguish between conditions and actions when drawing decision trees. This distinction is especially relevant when conditions and actions take place over a period of time and their sequence is important. For this purpose, use a square node to indicate an action and a circle to represent a condition. Using notation makes the decision tree more readable, as does numbering the circles and squares sequentially. Think of a circle as signifying IF, whereas the square means THEN. Figure 3.3 illustrates a decision tree example.

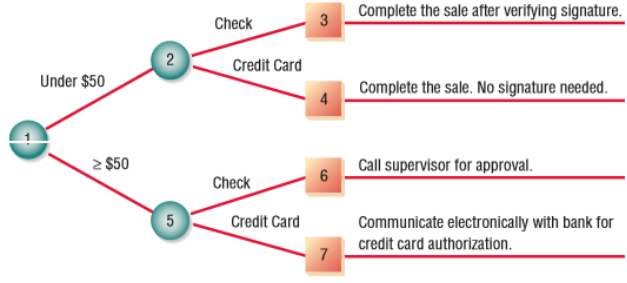


Figure 3.3: A decision Tree Example.

In drawing the tree:

1. Identify all conditions and actions and their order and timing (if they are critical).
2. Begin building the tree from left to right, making sure you list all possible alternatives before moving to the right.

The decision tree has three main advantages over a decision table.

- First, it takes advantage of the sequential structure of decision tree branches so that the order of checking conditions and executing actions is immediately noticeable.
- Second, conditions and actions of decision trees are found on some branches but not on others, which contrasts with decision tables, in which they are all part of the same table. Those conditions and actions that are critical are connected directly to other conditions and actions,

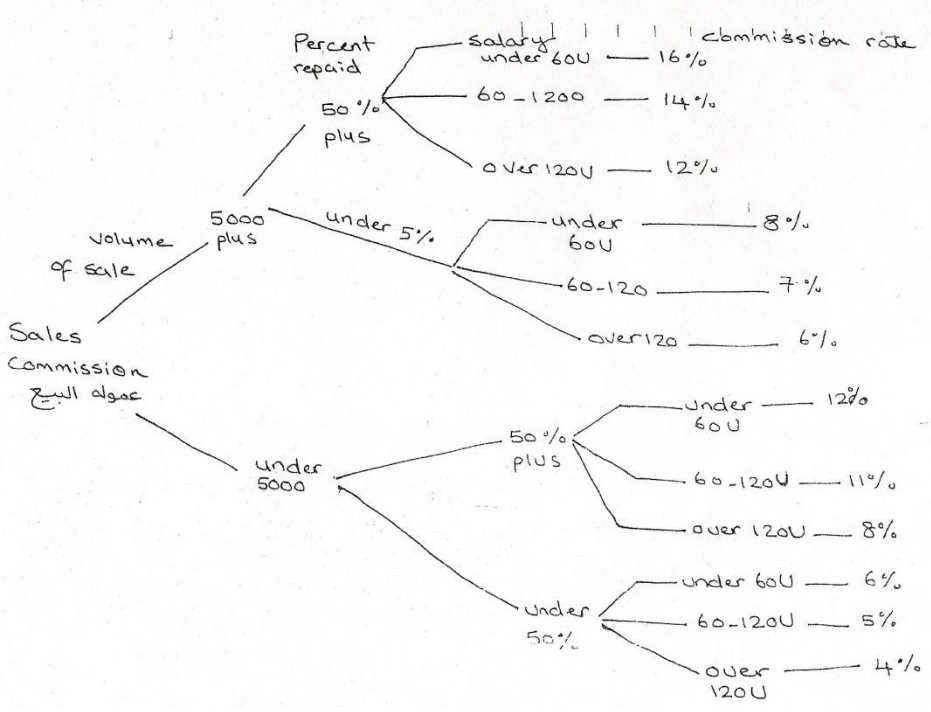
whereas those conditions that do not matter are absent. In other words, the tree does not have to be symmetrical.

- Third, compared with decision tables, decision trees are more readily understood by others in the organization. Consequently, they are more appropriate as a communication tool.

Example

Draw the decision tree for the decision table example in the previous section.

Answer



3.1.5 Choosing A Structures Decision Analysis Technique

We have examined the three techniques for analysis of structured decisions: structured English, decision tables, and decision trees. Although they need not be used exclusively, it is customary to choose one analysis technique for a decision rather than employing all three. The following

guidelines provide you with a way to choose one of the three techniques for a particular case:

1. Use structured English when
   a. There are many repetitious actions, OR
   b. Communication to end users is important.
2. Use decision tables when
   a. Complex combinations of conditions, actions, and rules are found, OR
   b. You require a method that effectively avoids impossible situations, redundancies, and contradictions.
3. Use decision trees when
   a. The sequence of conditions and actions is critical, OR
   b. When not every condition is relevant to every action (the branches are different).

3.2 Activty Planning and Control

Project management involves the general tasks of planning and control. ***Planning*** includes all the activities required to select a systems analysis team, assign members of the team to appropriate projects, estimate the time required to complete each task, and schedule the project so that tasks are completed in a timely fashion. ***Control*** means using feedback to monitor the project, including comparing the plan for the project with its actual evolution. In addition, control means taking appropriate action to expedite or reschedule activities to finish on time while motivating team members to complete the job properly.

3.3 Using Gantt Charts for Project Scheduling

It is a chart on which bars represent each task or activity. The length of each bar represents the relative length of the task. Figure 3.4 is an example of a two-dimensional Gantt chart in which time is indicated on the horizontal dimension and a description of activities makes up the vertical dimension. These activities overlap part of the time. Main advantages of Gantt chart is:

- its simplicity.
- It is an easy way to schedule tasks.
- It lends itself to worthwhile communication with end users.
- Bars representing activities or tasks are drawn to scale; the size of the bar indicates the relative length of time it will take to complete each task.

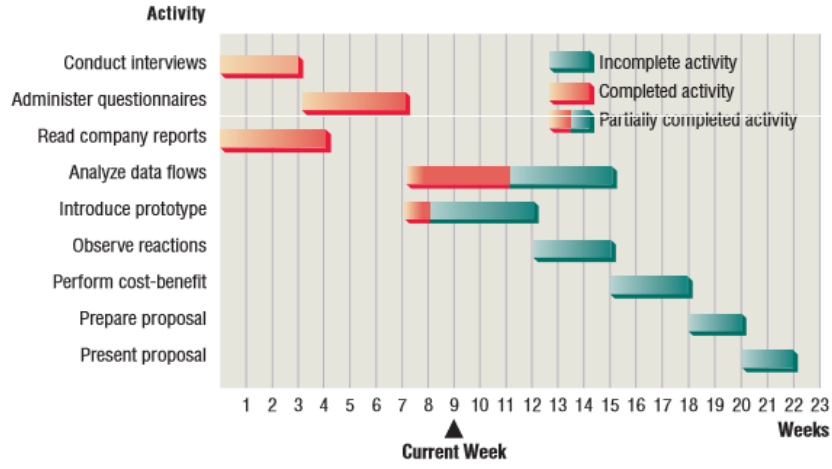


Figure 3.4: a two-dimensional Gantt chart for planning activities.

3.4 Using PERT Diagrams and Critical Path Management (CPM)

PERT is an acronym for ***Program Evaluation and Review Techniques***. A program (a synonym for a project) is represented by a network of nodes and arrows that are then evaluated to determine the critical activities, improve the schedule if necessary, and review progress once the project is undertaken. PERT is useful when activities can be done in parallel rather than in sequence. The systems analyst can benefit from PERT by applying it to systems projects on a smaller scale, especially when some team members can be working on certain activities at the same time that fellow members are working on other tasks.

Occasionally, PERT diagrams need pseudo-activities, referred to as dummy activities, to preserve the logic of or clarify the diagram, Figure 3.5.

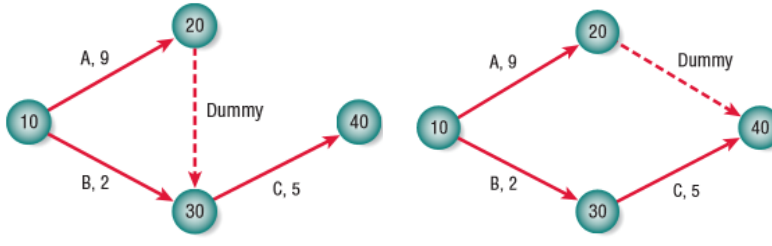


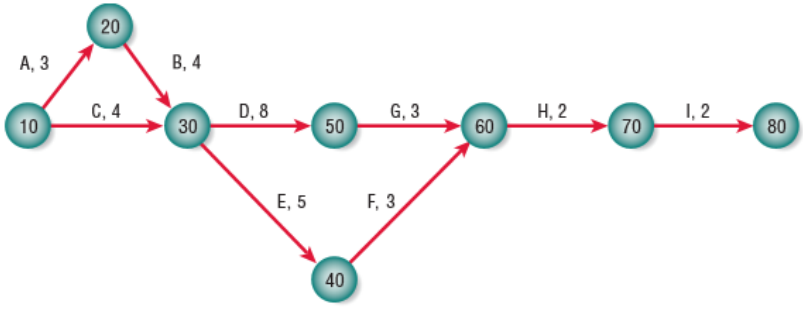
Figure 3.5: Dummy activities.

Therefore, there are many reasons for using a PERT diagram over a Gantt chart. The PERT diagram allows:

1. Easy identification of the order of precedence.
2. Easy identification of the critical path and thus critical activities.
3. Easy determination of slack time.

A PERT EXAMPLE.

Activity		Predecessor	Duration
A	Conduct interviews	None	3
B	Administer questionnaires	A	4
C	Read company reports	None	4
D	Analyze data flow	B, C	8
E	Introduce prototype	B, C	5
F	Observe reactions to prototype	E	3
G	Perform cost-benefit analysis	D	3
H	Prepare proposal	F, G	2
I	Present proposal	H	2



Exmple

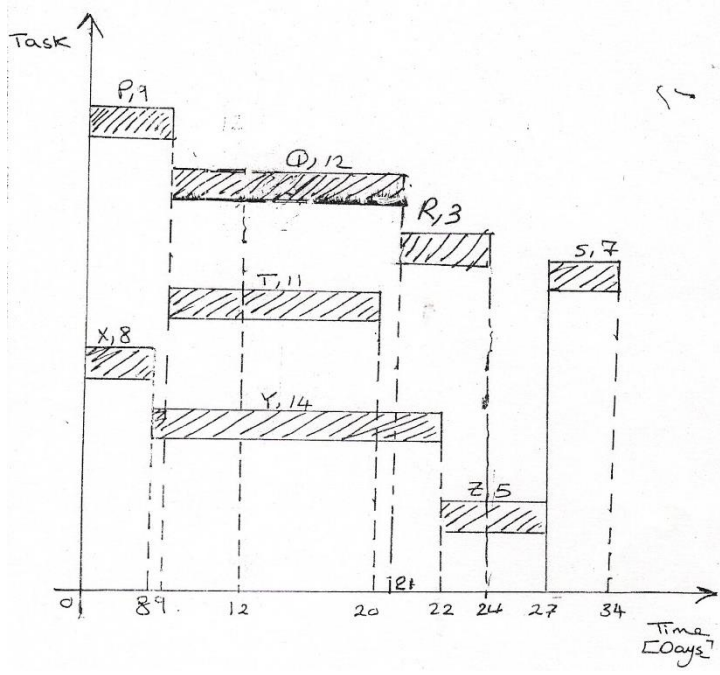
Given the following table:

Activity	Predecessor	Duration Time	Crash time	Cost/week
P	None	9	5	50$
Q	P	12	8	160$
R	Q	3	3	130$
S	R, Z	7	6	300$
T	P	11	9	80$
X	None	8	8	70$
Y	X	14	8	200$
Z	T, Y	5	2	150$

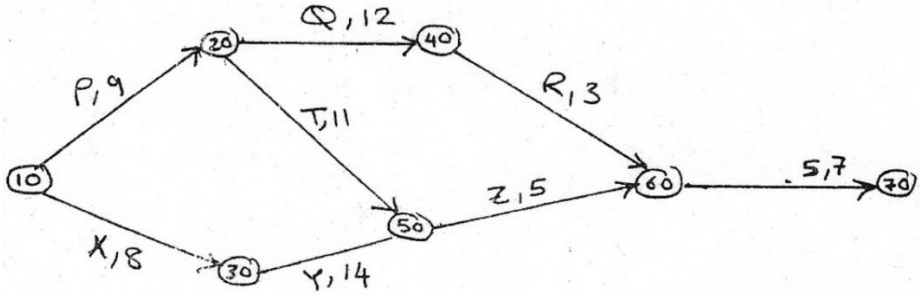
a) Draw Gantt Chart
b) Draw PERT Diagram
c) Find Different Pathes
d) Find the critical path (C.P)
e) Find the Comulitive Cost

Answer:

a) Gantt Chart:



b) PERT Diagram



c) Different Pathes

Path 1 (P1)

P – Q – R – S ➔ 9 + 12 + 3 +7 = 31 days

Path 2 (P2)

P – T – Z – S ➔ 9 + 11 + 5 + 7 = 32 days

Path 3 (P3)

X – Y – Z – S ➔ 8 + 14 + 5 + 7 = 34 dayes

d) The Crtical Path (C.P)

Critical path is the longest path which is (P3) = 34 days

e) The Comulitive Cost

Eligile activities	Activity Chosen	Time for each Path			Cost	Comulitive Cost
		P1 31	P2 32	P3 34		
[Y] or [Z] or [S]	Z	31	31	33	150$	150$
[Y] or [Z] or [S]	Z	31	30	32	150$	300$
[Y] or [Z] or [S]	Z	31	29	31	150$	450$
[S] or [P&Y]	[P & Y]	30	28	30	50$+200$	700$
[S] or [P&Y]	[P & Y]	29	27	29	50$+200$	950$
[S] or [P&Y]	[P & Y]	28	26	28	50$+200$	1200$
[S] or [P&Y]	[P & Y]	27	25	27	50$+200$	1450
[S] or [P&Y]	S	26	24	26	300$	1750$
[Q & Y]	[Q & Y]	25	24	25	160$ + 200$	2110
[Q & Y]	[Q & Y]	24	24	24	160$+200$	2470$

.

CHAPTER 4
THE ANALYSIS PROCESS

4.1 Using Data Flow Diagrams (DFD)

DFDs emphasize the processing of data or the transforming of data as they move through a variety of processes. They focus on the data flowing into and out of the system and the processing of the data. In logical DFDs, there is no distinction between manual or automated processes. Neither are the processes graphically depicted in chronological order. Rather, processes are eventually grouped together if further analysis dictates that it makes sense to do so. Manual processes are put together, and automated processes can also be paired with each other. This concept, called partitioning.

Conventions Used in Data Flow Diagrams Four basic symbols are used to chart data movement on data flow diagrams: a double square, an arrow, a rectangle with rounded corners, and an open-ended rectangle (closed on the left side and open ended on the right), as shown in Figure 4.1.

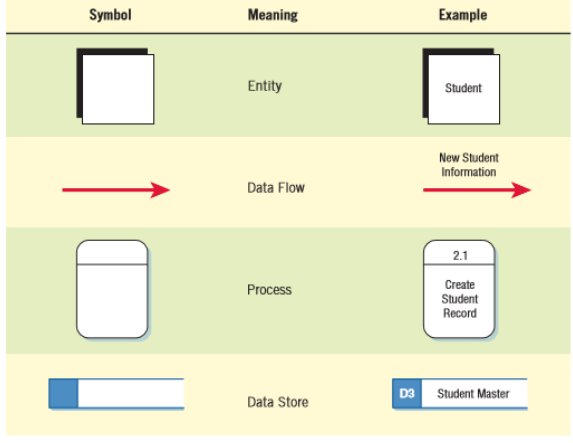


Figure 4.1: The four basic symbols used in data flow diagrams.

The double square is used to depict an ***external entity*** (another department, a business, a person, or a machine) that can send data to or receive data from the system. The external entity, or just entity, is also called a source or destination of data, and it is considered to be external to the system being described. Each entity is labeled with an appropriate name. Although it interacts with the system, it is considered as outside the boundaries of the system. Entities should be named with a noun. The same entity may be used more than once on a given data flow diagram to avoid crossing data flow lines.

The arrow shows movement of data from one point to another, with the head of the arrow pointing toward the data's destination. ***Data flows*** occurring

simultaneously can be depicted doing just that through the use of parallel arrows. Because an arrow represents data about a person, place, or thing, it too should be described with a noun.

Arectangle with rounded corners is used to show the occurrence of a transforming process. ***Processes*** always denote a change in or transformation of data; hence, the data flow leaving a process is always labeled differently than the one entering it.

Processes represent work being performed in the system and should be named using one of the following formats. Aclear name makes it easier to understand what the process is accomplishing.

1. When naming a high-level process, assign the process the name of the whole system. An example is INVENTORY CONTROL SYSTEM.
2. When naming a major subsystem, use a name such as INVENTORY REPORTING SUBSYSTEM or INTERNET CUSTOMER FULFILLMENT SYSTEM.
3. When naming detailed processes, use a verb-adjective-noun combination. The verb describes the type of activity, such as COMPUTE, VERIFY, PREPARE, PRINT, or ADD. The noun indicates what the major outcome of the process is, such as REPORT or RECORD. The adjective illustrates which specific output, such as BACKORDERED or INVENTORY, is produced.

Aprocess must also be given a unique identifying number indicating its level in the diagram. Several data flows may go into and out of each process. Examine processes with only a single flow in and out for missing data flows.

The last basic symbol used in data flow diagrams is an open-ended rectangle, which represents a ***data store***. These symbols are drawn only wide enough to allow identifying lettering between the parallel lines.

In logical data flow diagrams, the type of physical storage is not specified. At this point the data store symbol is simply showing a depository for data that allows examination, addition, and retrieval of data. The data store may represent amanual store, such as afiling cabinet, or a computerized file or database. Because data stores represent a person, place, or thing, they are named with a noun.Temporary data stores, such as scratch paper or a temporary computer file, are not included on the data flow diagram. Give each data store a unique reference number, such as D1, D2, D3, and so on.

4.1.1 Developing Data Flow Diagrams

Data flow diagrams can and should be drawn systematically. First, the systems analyst needs to conceptualize data flows from a top-down perspective. To begin a data flow diagram, collapse the organization's system narrative (or story) into a list with the four categories of external entity, data flow, process, and data store. This list in turn helps determine the boundaries of the system you will be describing. Once a basic list of data elements has been compiled, begin drawing a context diagram. Here are a few basic rules to follow:

1. The data flow diagram must have at least one process, and must not have any freestanding objects or objects connected to themselves.
2. A process must receive at least one data flow coming into the process and create at least one data flow leaving from the process.
3. A data store should be connected to at least one process.
4. External entities should not be connected to each other. Although they communicate independently, that communication is not part of the system we design using DFDs.

4.1.2 Creating the Context Diagram

The first model is the context-level data flow diagram (also called an environmental model). The context-level data flow diagram is one way to show the scope of the system, or what is to be included in the system. The external entities are outside of the scope and something over which the system has no control.

The context diagram is the highest level in a data flow diagram and contains only one process, representing the entire system. The process is given the number zero. All external entities are shown on the context diagram, as well as major data flow to and from them. The diagram does not contain any data stores and is fairly simple to create, once the external entities and the data flow to and from them are known to analysts.

As shown in Figure 4.2, the context-level data flow diagram employs only three symbols: (1) a rectangle with rounded corners, (2) a square with two shaded edges, and (3) an arrow. Processes transform incoming data into outgoing information, and the content level has only one process, representing the entire system. The external entity represents any entity that supplies or

receives information from the system but is not a part of the system. This entity may be a person, a group of people, a corporate position or department, or other systems. Lines that connect the external entities to the process are called data flows, and they represent data.

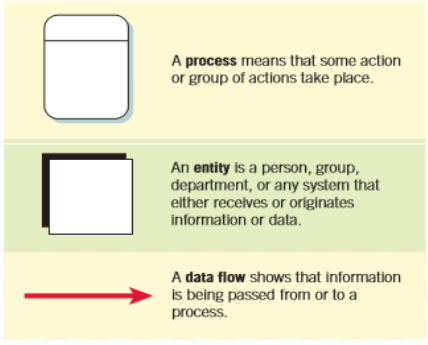


Figure 4.2: The basic symbols of a data flow diagram.

An example of a context-level data flow diagram is found in Figure 4.3. The passenger (an entity) initiates a travel request (data flow). The context-level diagram doesn't show enough detail to indicate exactly what happens but we can see that the passenger's preferences and the available flights are sent to the travel agent, who sends ticketing information back to the process. We can also see that the passenger reservation is sent to the airline. The context-level data flow diagram serves as a good starting point for drawing the use case diagram.

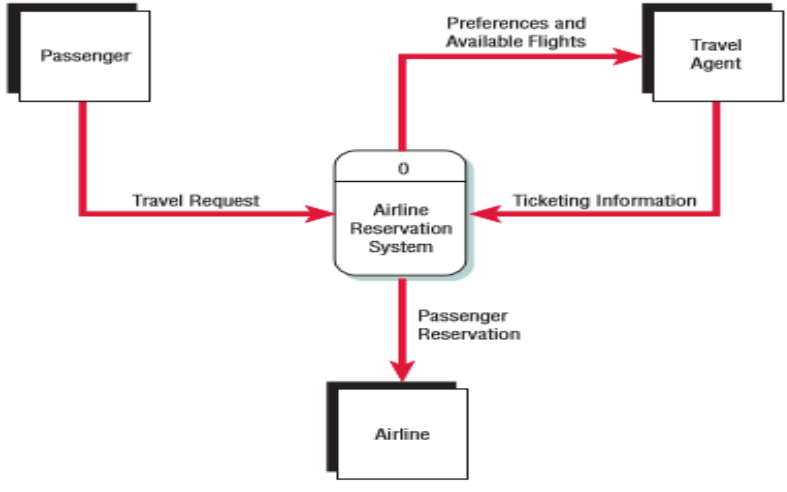


Figure 4.3: An airline reservation system context-level data flow diagram.

4.1.3 Drawing Diagram 0 (The Next Level)

More detail than the context diagram permits is achievable by "***exploding the diagrams.***" Inputs and outputs specified in the first diagram remain constant in all subsequent diagrams. The rest of the original diagram, however, is exploded into close-ups involving three to nine processes and showing data stores and new lower-level data flows. The effect is that of taking a magnifying glass to view the original data flow diagram. Each exploded diagram should use only a single sheet of paper. By exploding DFDs into subprocesses, the systems analyst begins to fill in the details about data movement. The handling of exceptions is ignored for the first two or three levels of data flow diagramming. Diagram 0 is the explosion of the context diagram and may include up to nine processes. Including more processes at this level will result in a cluttered diagram that is difficult to understand. Each process is numbered with an integer, generally starting from the upper left-hand corner of the diagram and working toward the lower right-hand corner. The major data stores of the system (representing master files) and all external entities are included on Diagram 0. Figure 4.4 schematically illustrates both the context diagram and Diagram 0. Because a data flow diagram is two-dimensional (rather than linear), you may start at any point and work forward or backward through the diagram. If you are unsure of what you would include at any point, take a different external entity, process, or data store, and then start drawing the flow from it.

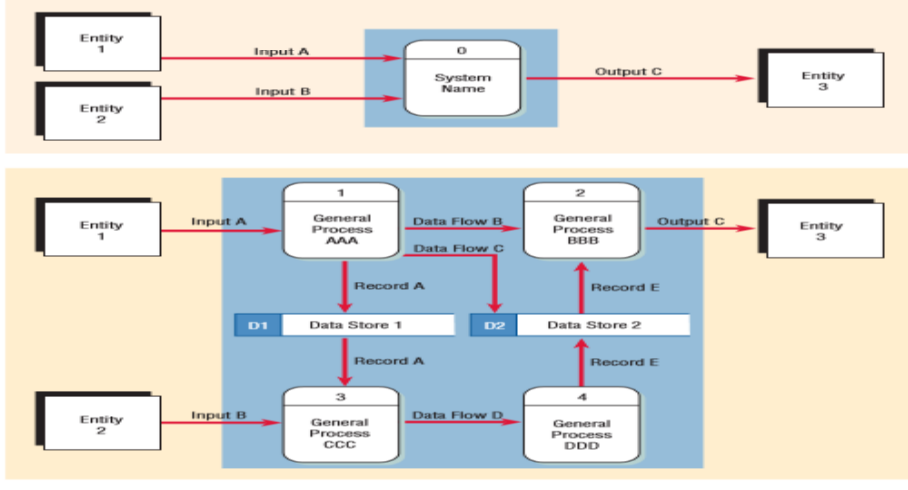


Figure 4.4: Context diagrams (above) can be "exploded" into Diagram 0 (below).

4.1.4 Creating Child Diagrams (More Detailed Levels)

Each process on Diagram 0 may in turn be exploded to create a more detailed child diagram. The process on Diagram 0 that is exploded is called the parent process, and the diagram that results is called the child diagram.The primary rule for creating child diagrams, vertical balancing, dictates that a child diagram cannot produce output or receive input that the parent process does not also produce or receive. All data flow into or out of the parent process must be shown flowing into or out of the child diagram. The child diagram is given the same number as its parent process in Diagram 0. For example, process 3 would explode to Diagram 3. The processes on the child diagram are numbered using the parent process number, a decimal point, and a unique number for each child process. Entities are usually not shown on the child diagrams below Diagram 0. Data flow that matches the parent flow is called an ***interface data flow*** and is shown as an arrow from or into a blank area of the child diagram. If the parent process has data flow connecting to a data store, the child diagram may include the data store as well. In addition, this lower-level diagram may contain data stores not shown on the parent process. Minor data flow, such as an error line, may be included on a child diagram but not on the parent.

Processes may or may not be exploded, depending on their level of complexity. When a process is not exploded, it is said to be ***functionally primitive*** and is called a ***primitive process***. Logic is written to describe these processes. Figure 4.5 illustrates detailed levels in a child data flow diagram.

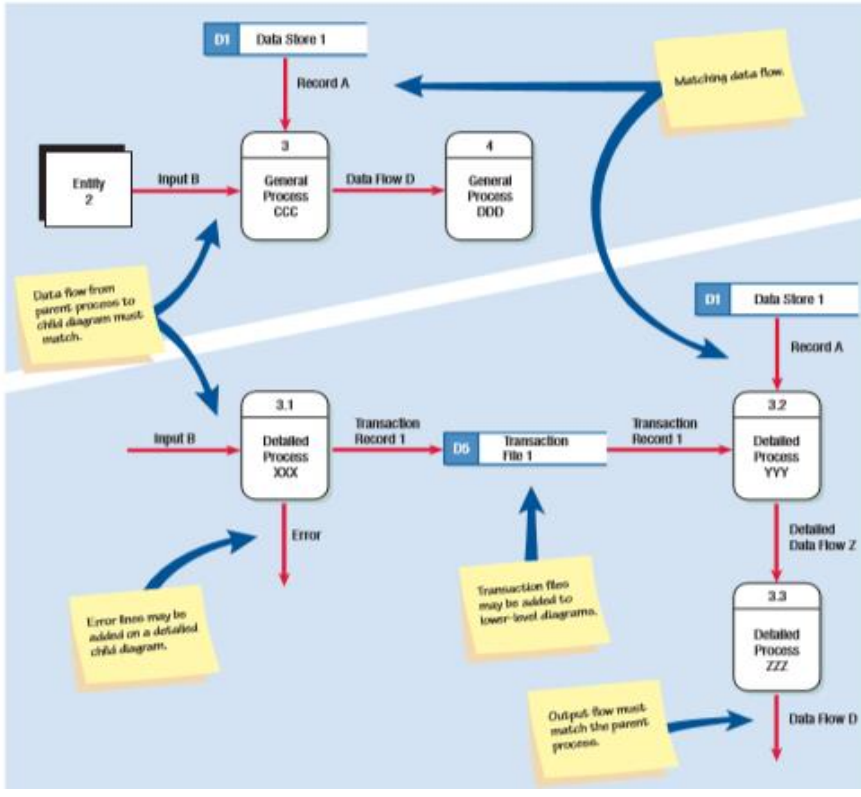


Figure 4.5: Differences between the parent diagram and the child diagram.

4.1.5 Checking the Diagrams for Errors

Several common errors made when drawing data flow diagrams are as follows, Figure 4.6:

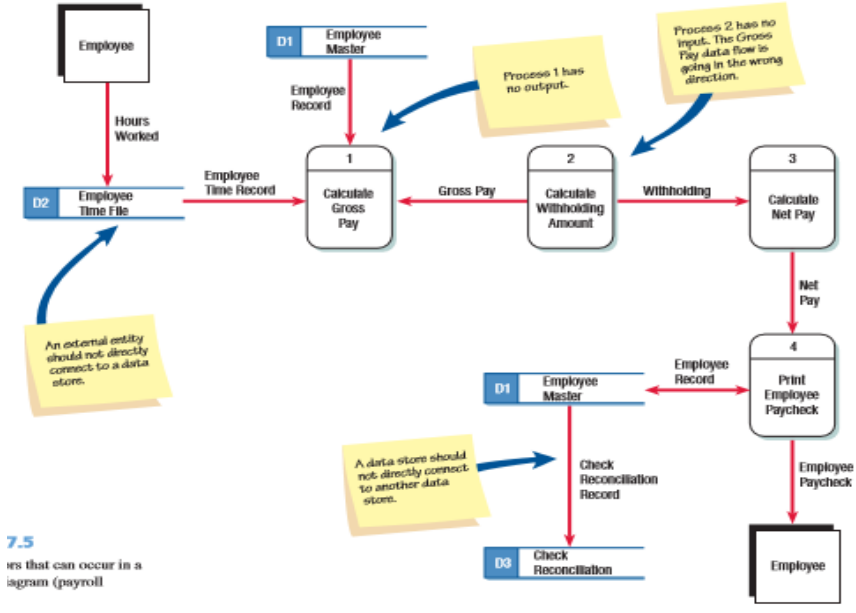


Figure 4.6: Typical errors that can occur in a data flow diagram.

1. Forgetting to include a data flow or pointing an arrow in the wrong direction. An example is a drawn process showing all its data flow as either input or output. Each process transforms data and must receive input and produce output.
2. Connecting data stores and external entities directly to each other; they must connect only with a process.
3. Incorrectly labeling processes or data flow. Inspect the data flow diagram to ensure that each object or data flow is properly labeled. Aprocess should indicate the system name or use the verb-adjective-noun format. Each data flow should be described with a noun.
4. Including more than nine processes on a data flow diagram. If more than nine processes are involved in a system, group some of the processes that work together into a subsystem and place them in a child diagram.
5. Omitting data flow. Examine your diagram for linear flow, that is, data flow in which each process has only one input and one output. Except in the case of very detailed child data flow diagrams, linear data flow

is somewhat rare. Its presence usually indicates that the diagram has missing data flow.

6. Creating unbalanced decomposition (or explosion) in child diagrams. Each child diagram should have the same input and output data flow as the parent process. An exception to this rule is minor output, such as error lines, which are included only on the child diagram.

4.1.6 Logical and Physical Data Flow Diagrams

Data flow diagrams are categorized as either logical or physical. A ***logical data flow diagram*** focuses on the business and how the business operates. It is not concerned with how the system will be constructed. Instead, it describes the business events that take place and the data required and produced by each event. Conversely, a ***physical data flow diagram*** shows how the system will be implemented, including the hardware, software, files, and people involved in the system. Notice that the logical model reflects the business, whereas the physical model depicts the system. Ideally, systems are developed by analyzing the current system (the current logical DFD) and then adding features that the new system should include (the proposed logical DFD). Finally, the best methods for implementing the new system should be developed (the physical DFD). This progression is shown in Figure 4.7. Developing a logical data flow diagram for the current system affords a clear understanding of how the current system operates, and thus a good starting point for developing the logical model of the current system.

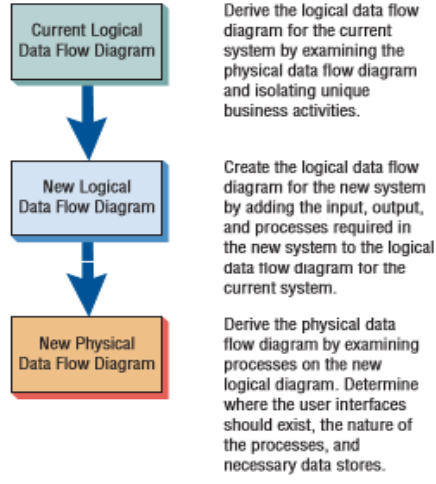


Figure 4.7: The progression of models from logical to physical.

Figure 4.8 shows a logical data flow diagram and a physical data flow diagram for a grocery store cashier.

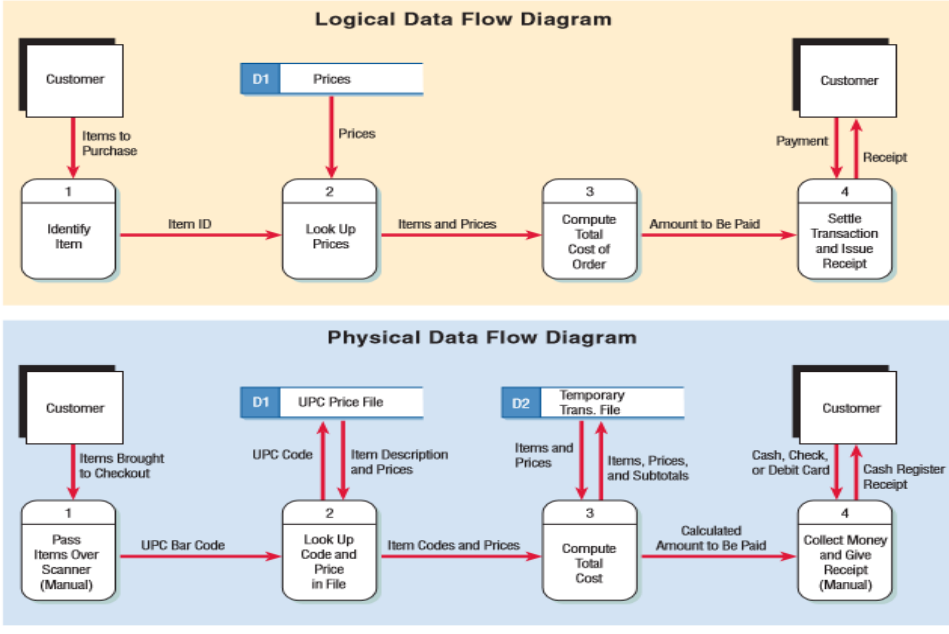


Figure 4.8: The physical and logical data flow diagrams.

A logical model is easier to use when communicating with users of the system because it is centered on business activities. Users will thus be familiar with the essential activities and many of the human information requirements of each activity. Systems formed using a logical data flow diagram are often more stable because they are based on business events and not on a particular technology or method of implementation. Logical data flow diagrams represent features of a system that would exist no matter what the physical means of doing business are.

4.2 Systems and the Entity–Relationship Model

Another way a systems analyst can show the scope of the system and define proper system boundaries is to use an entity-relationship model. The elements that make up an organizational system can be referred to as entities. An entity may be a person, a place, or a thing, such as a passenger on an airline, a destination, or a plane. Alternatively, an entity may be an event, such as the

end of the month, a sales period, or a machine breakdown. A relationship is the association that describes the interaction among the entities. There are many different conventions for drawing entity-relationship (E-R) diagrams (with names like crow's foot, Arrow, or Bachman notation).

4.2.1 Relationships.

Relationships are associations between entities (sometimes they are referred to as data ***associations***). Figure 4.9 is an entity-relationship (E-R) diagram that shows various types of relationships.

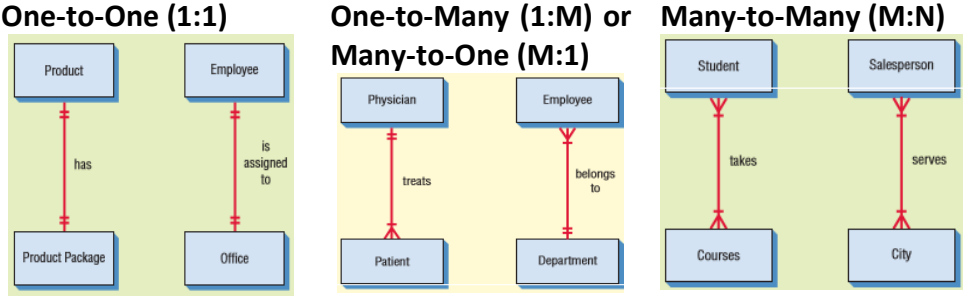


Figure 4.9: Entity-relationship (E-R) diagrams.

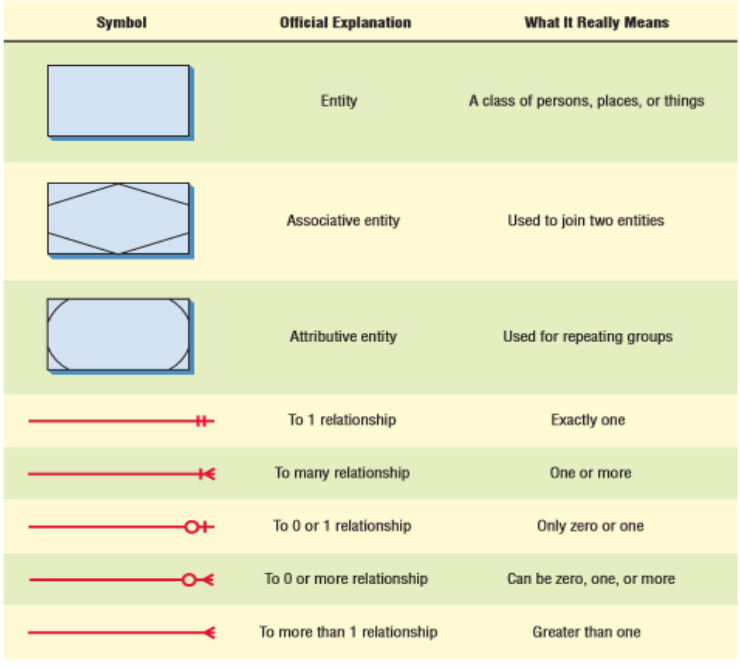
The first type of relationship is a one-to-one relationship (designated as 1:1). Another type of relationship is a one-to-many (1:M) or a many-to-one association. Finally, a many-to-many relationship (designated as M:N) describes the possibility that entities may have many associations in either direction.

The standard symbols for ***crow's foot notation***, the official explanation of the symbols, and what they actually mean, are all given in Figure 4.10.

Notice that the symbol for an entity is a rectangle. An ***entity*** is defined as a class of a person, place, or thing. A rectangle with a diamond inside stands for ***an associative entity***, which is used to join two entities. A rectangle with an oval in it stands for an ***attributive entity***, which is used for repeating groups. The other notations necessary to draw E-R diagrams are the connections, of which there are five different types. In the lower portion of the figure, the meaning of the notation is explained.

When a straight line connects two plain entities and the ends of the line are both marked with two short marks (||), a one-to-one relationship exists. Following that you will notice a crow's foot with a short mark (|); when this notation links entities, it indicates a relationship of one-to-one or oneto-many

(to one or more). Entities linked with a straight line plus a short mark (|) and a zero (which looks more like a circle, O) are depicting a relationship of one-to-zero or one-to-one (only zero or one). A fourth type of link for relating entities is drawn with a straight line marked on the end with a zero (O) followed by a crow's foot. This type shows a zero-to-zero, zero-to-one, or zero-to-many relationship. Finally, a straight line linking entities with a crow's foot at the end depicts a relationship to more than one.



Symbol	Official Explanation	What It Really Means
	Entity	A class of persons, places, or things
	Associative entity	Used to join two entities
	Attributive entity	Used for repeating groups
	To 1 relationship	Exactly one
	To many relationship	One or more
	To 0 or 1 relationship	Only zero or one
	To 0 or more relationship	Can be zero, one, or more
	To more than 1 relationship	Greater than one

Figure 4.10: The entity-relationship symbols and their meanings.

An entity may have a relationship connecting it to itself. This type of relationship is called a ***self-join relationship***; the implication is that there must be a way to link one record in a file to another record in the same file. The relationships in words can be written along the top or the side of each connecting line. In practice, you see the relationship in one direction, although you can write relationships on both sides of the line, each representing the point of view of one of the two entities. Figure 4.11 lists a number of typical entity relationships.

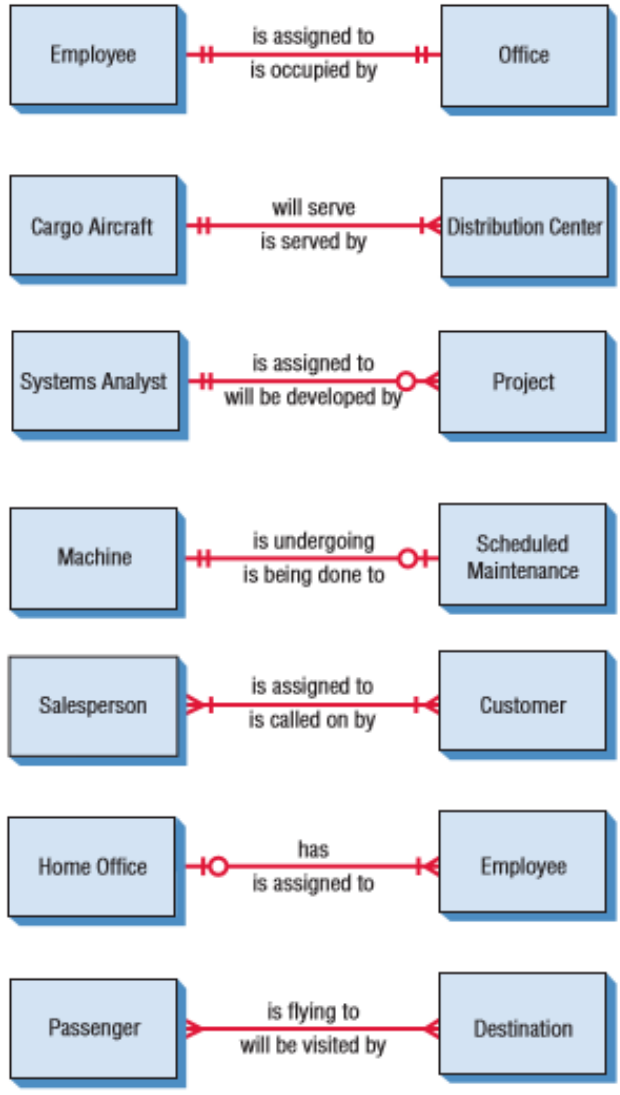


Figure 4.11: Examples of different types of relationships in E-R diagrams.

Up to now we have modeled all our relationships using just one simple rectangle and a line. This method works well when we are examining the relationships of real things such as real people, places, and things. Sometimes, though, we create new items in the process of developing an information system. Some examples are invoices, receipts, files, and databases. When we want to describe how a person relates to a receipt, for example, it becomes

convenient to indicate the receipt in a different way, as shown in Figure 23 as an associative entity. An ***associative entity*** can only exist if it is connected to at least two other entities. For that reason, some call it a gerund, a junction, an intersection, or a concatenated entity. This wording makes sense because a receipt wouldn't be necessary unless there were a customer and a salesperson making the transaction. Another type of entity is the ***attributive***. When an analyst wants to show data that are completely dependent on the existence of a fundamental entity, an attributive entity should be used.

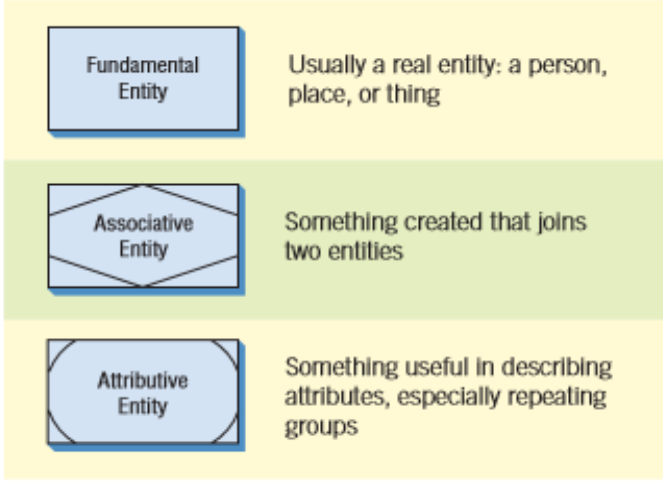
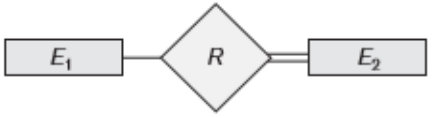


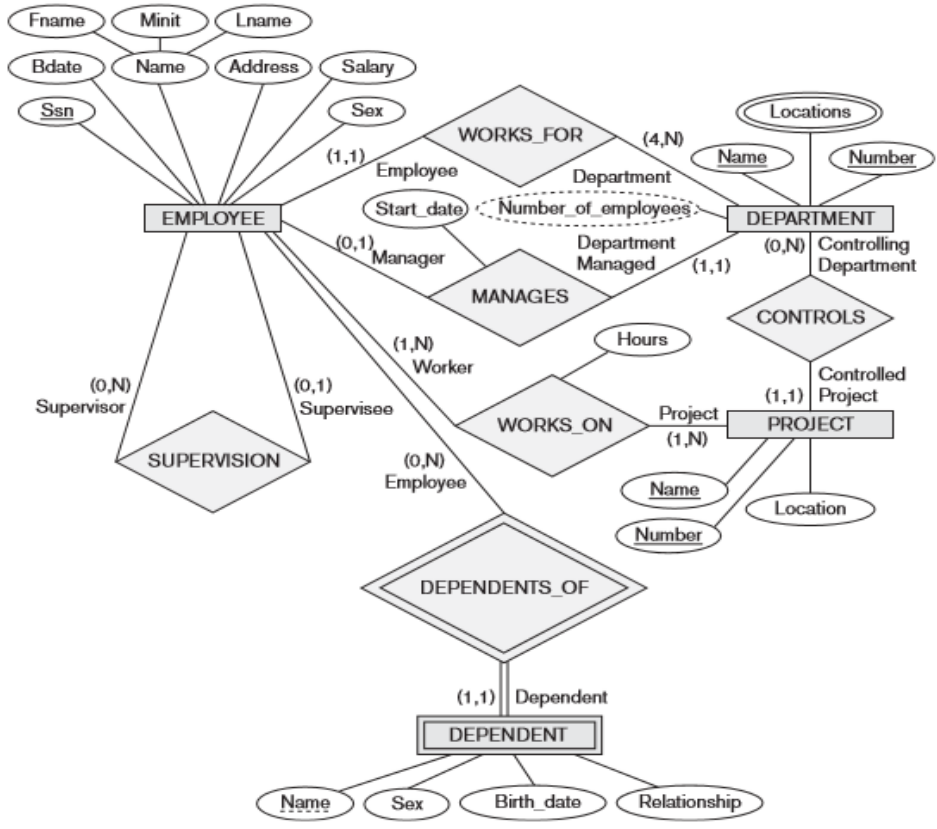
Figure 4.12: Three different types of entities used in E-R diagrams.

In sketching out some basic E-R diagrams, the analyst needs to:

1. List the entities in the organization to gain a better understanding of the organization.
2. Choose key entities to narrow the scope of the problem to a manageable and meaningful dimension.
3. Identify what the primary entity should be.
4. Confirm the results of steps 1 through 3 through other data-gathering methods.

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for E_1:E_2 in R
	Structural Constraint (min, max) on Participation of E in R

The following diagram refers to an ER diagram for the company schema with structural constraints specified using (min, max) notation and role names.



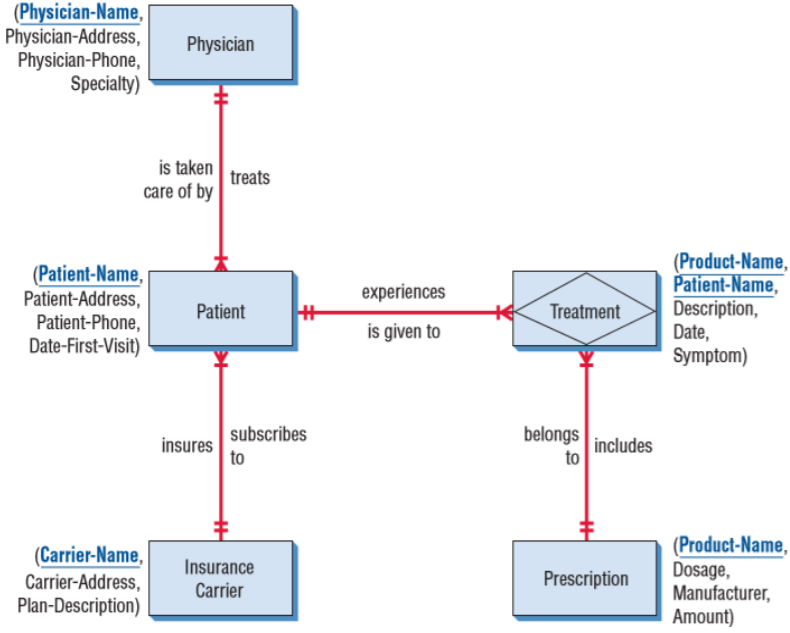
The following diagram draws an Entity Relationship Diagram (ERD) for patient treatment in a billing system showing entities, relationships, attributes, and keys.

The entities are PRESCRIPTION, PHYSICIAN, PATIENT, and INSURANCE CARRIER. The entity TREATMENT is not important for the billing system, but it is part of the E-R diagram because it is used to bridge the gap between PRESCRIPTION and PATIENT.

A PHYSICIAN treats many PATIENT(s), who each subscribe to an individual INSURANCE CARRIER. The PATIENT is only one of many patients that subscribe to that particular INSURANCE CARRIER.

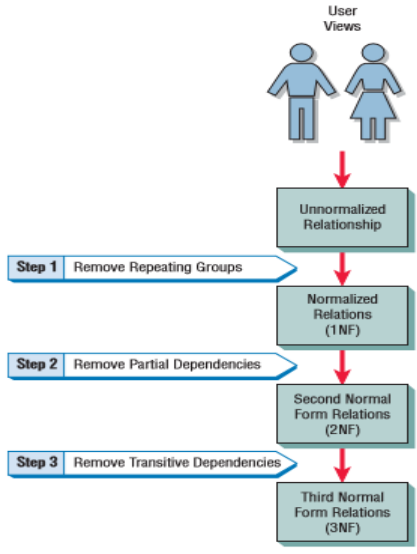
To complete the PHYSICIAN's records, the physician needs to keep information about the treatments a PATIENT has. Many PATIENT(s) experience many TREATMENT(s), making it a many-to-many (M:N) relationship. TREATMENT(s) can include the taking of PRESCRIPTION(s), as many treatments may call for combinations of pharmaceuticals and many drugs may work for many treatments.

The entity PHYSICIAN has a PHYSICIAN-NAME, ADDRESS, PHONE, and SPECIALTY. The PATIENT entity has a PATIENT-NAME, ADDRESS, PHONE, AND DATE-FIRST-VISIT. The entity INSURANCE CARRIER Has a CARRIER-NAME, ADDRESS, PLAN-DESCRIPTION, The TREATMENT entity has a PRODUCT-NAME, PATIENT-NAME, DESCRIPTION, DATE, and SYMPOTM. The entity PRESCRIPTION has a PRODUCT-NAME, DOSAGE, MANUFACTURER, and AMOUNT.



4.3 Database Design: Normalization

Normalization is the transformation of complex user views and data stores to a set of smaller, stable data structures. A data structure is normalized in three steps, as shown in the following figure. Each step involves an important procedure, one that simplifies the data structure. The relation derived from the user view or data store will most likely be unnormalized.



Normalization of a relation is accomplished in three major steps.

The first stage of the process includes ***removing all repeating groups*** and ***identifying the primary key.*** To do so, the relation needs to be broken up into two or more relations. At this point, the relations may already be of the third normal form, but it is likely more steps will be needed to transform the relations to the third normal form. The second step ensures that ***all nonkey attributes are fully dependent on the primary key***. ***All partial dependencies are removed*** and placed in another relation. The third step ***removes any transitive dependencies.*** A transitive dependency is one in which nonkey attributes are dependent on other nonkey attributes.

- ***FIRST NORMAL FORM (1NF).*** The first step in normalizing a relation is to remove the repeating groups.
- ***SECOND NORMAL FORM (2NF).*** In the second normal form, all the attributes will be functionally dependent on the primary key.
- ***THIRD NORMAL FORM (3NF).*** Anormalized relation is in the third normal form if all the nonkey attributes are fully functionally dependent on the primary key and there are no transitive (nonkey) dependencies.

The third normal form is adequate for most database design problems. The simplification gained from transforming an unnormalized relation into a set of 3NF relations is a tremendous benefit when it comes time to insert, delete, and update information in the database.

CHAPTER 5
UML CONCEPTS AND DIAGRAMS

5.1 Use Case Modeling

Originally introduced as a diagram for use in object-oriented UML, use cases are now being used regardless of the approach to systems development. It can be used as part of the SDLC or in agile modeling. The word use is pronounced as a noun (yoos) rather than a verb (yooz). A use case model describes what a system does without describing how the system does it; that is, it is a logical model of the system.

The use case model reflects the view of the system from the perspective of a user outside of the system (i.e., the system requirements). An analyst develops use cases in a cooperative effort with the business experts who help define the requirements of the system. The use case model provides an effective means of communication between the business team and the development team.

A use case model partitions the way the system works into behaviors, services, and responses (the use cases) that are significant to the users of the system. From the perspective of an actor (or user), a use case should produce something that is of value. Therefore, the analyst must determine what is important to the user, and remember to include it in the use case diagram. For example, is entering a password something of value to the user? It may be included if the user has a concern about security or if it is critical to the success of the project.

5.1.1 Use Case Symbols

A use case provides developers with a view of what the users want. It is free of technical or implementation details. We can think of a use case as a sequence of transactions in a system. The use case model is based on the interactions and relationships of individual use cases. A use case always describes three things: an ***actor*** that initiates an event; the ***event*** that triggers a use case; and the ***use case*** that performs the actions triggered by the event. In a use case, an actor using the system initiates an event that begins a related series of interactions in the system. Use cases are used to document a single transaction or event. An event is an input to the system that happens at a specific time and place and causes the system to do something.

A use case diagram contains the actor and use case symbols, along with connecting lines. Actors are similar to external entities; they exist outside of

the system. The term actor refers to a particular role of a user of the system. Even though it is the same person in the real world, it is represented as two different symbols on a use case diagram, because the person interacts with the system in different roles. The actor exists outside of the system and interacts with the system in a specific way. An actor can be a human, another system, or a device such as a keyboard or Web connection. Actors can initiate an instance of a use case. An actor may interact with one or more use cases, and a use case may involve one or more actors.

Actors may be divided into two groups. ***Primary actors*** supply data or receive information from the system. Some users directly interact with the system (system actors), but primary actors may also be business people who do not directly interact with the system but have a stake in it. Primary actors are important because they are the people who use the system and can provide details on what the use case should do. They can also provide a list of goals and priorities. ***Supporting actors*** (also called ***secondary actors***) help to keep the system running or provide other services. These are the people who run the help desk, the analysts, programmers, and so on.

It is better to create fewer use cases rather than more. Often queries and reports are not included; 20 use cases (and no more than 40 or 50) are sufficient for a large system. Use cases may also be nested, if needed. Some use cases use the verb manage to group use cases for adding, deleting, and changing into another, lower-level, use case diagram. You can include a use case on several diagrams, but the actual use case is defined only once in the repository. A use case is named with a verb and a noun.

5.1.2 Use Case Relationships

Active relationships are referred to as behavioral relationships and are used primarily in use case diagrams. There are four basic types of behavioral relationships: communicates, includes, extends, and generalizes. Notice that all these terms are action verbs. Figure 5.1 shows the arrows and lines used to diagram each of the four types of behavioral relationships.

COMMUNICATES. The behavioral relationship communicates is used to connect an actor to a use case. Remember that the task of the use case is to give some sort of result that is beneficial to the actor in the system.

INCLUDES. The includes relationship (also called uses relationship) describes the situation in which a use case contains behavior that is common

to more than one use case. In other words, the common use case is included in the other use cases. A dotted arrow that points to the common use case indicates the includes relationship.

Relationship	Symbol	Meaning
Communicates		An actor is connected to a use case using a line with no arrowheads.
Includes	<< include >>	A use case contains a behavior that is common to more than one other use case. The arrow points to the common use case.
Extends	<< extend >>	A different use case handles exceptions from the basic use case. The arrow points from the extended to the basic use case.
Generalizes		One UML "thing" is more general than another "thing." The arrow points to the general "thing."

Figure 5.1: Types of Behavioral Relationships.

EXTENDS. The extends relationship describes the situation in which one use case possesses the behavior that allows the new use case to handle a variation or exception from the basic use case. The arrow goes from the extended to the basic use case.

GENERALIZES. The generalizes relationship implies that one thing is more typical than the other thing. This relationship may exist between two actors or two use cases. The arrow points to the general thing.

5.1.3 Developing System Scope

The scope of a system defines its boundaries, what is in scope—or inside the system—and what is out of scope. The project usually has a budget that helps to define scope, and a start and end time. Actors are always outside the scope of the system. The communicates lines that connect actors to the use cases are the boundaries, and define the scope. Since a use case diagram is created early in the systems life cycle, the budget, starting time, and ending time may change as the project progresses; as the analyst learns more about the system, the use case diagrams, use case, and scope may change.

5.1.4 Developing Use Case Diagrams

The primary use case consists of a standard flow of events in the system that describes a standard system behavior. The primary use case represents the normal, expected, and successful completion of the use case. When diagramming a use case, start by asking the users to list everything the system should do for them. The analyst may also use agile stories sessions to develop

use cases. Write down who is involved with each use case, and the responsibilities or services the use case must provide to actors or other systems. In the initial phases, this may be a partial list that is expanded in the later analysis phases. Use the following guidelines:

1. Review the business specifications and identify the actors involved.
2. Identify the high-level events and develop the primary use cases that describe those events and how the actors initiate them. Carefully examine the roles played by the actors to identify all the possible primary use cases initiated by each actor. Use cases with little or no user interaction do not have to be shown.
3. Review each primary use case to determine the possible variations of flow through the use case. From this analysis, establish the alternative paths. Because the flow of events is usually different in each case, look for activities that could succeed or fail. Also look for any branches in the use case logic in which different outcomes are possible.

If a context-level data flow diagram has been created, it can be a starting point for creating a use case. The external entities are potential actors. Then examine the data flow to determine if it would initiate a use case or be produced by a use case.

5.1.5 Developing Use Case Scenarios

Each use case has a description. We will refer to the description as a use case scenario. As mentioned, the primary use case represents the standard flow of events in the system, and alternative paths describe variations to the behavior.

There is no standardized use case scenario format, so each organization is faced with specifying what standards should be included. Often the use cases are documented using a use case document template predetermined by the organization, which makes the use cases easier to read and provides standardized information for each use case in the model.

5.2 Class Diagram

Each class should have a name that differentiates it from all other classes. Class names are usually nouns or short phrases and begin with an uppercase letter. In UML, a class is drawn as a rectangle, Figure 5.2. The rectangle contains two other important features: a list of attributes and a series of

methods. These items describe a class, the unit of analysis that is a large part of what we call object-oriented analysis and design. An attribute describes some property that is possessed by all objects of the class. The class name is centered at the top of the class, usually in boldface type The area directly below the name shows the attributes, and the bottom portion lists the methods.

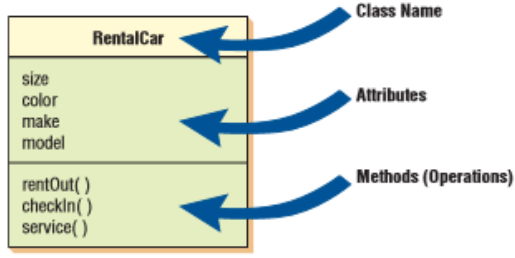


Figure 5.2: An example of a UML class.

Because programming occurs at the class level, defining classes is one of the most important object-oriented analysis tasks. Class diagrams show the static features of the system and do not represent any particular processing. A class diagram also shows the nature of the relationships between classes. ***Classes*** are represented by a rectangle on a class diagram. In the simplest format, the rectangle may include only the class name, but may also include the attributes and methods. ***Attributes*** are what the class knows about characteristics of the objects, and ***methods*** (also called operations) are what the class knows about how to do things. ***Methods*** are small sections of code that work with the attributes. Figure 5.4 illustrates a class diagram for course offerings.

The class diagram shows data storage requirements as well as processing requirements. The attributes (or properties) are usually designated as ***private***, or only available in the object. This is represented on a class diagram by a minus sign in front of the attribute name. Attributes may also be ***protected***, indicated with a pound symbol (#). These attributes are hidden from all classes except immediate subclasses. Under rare circumstances, an attribute is ***public***, meaning that it is visible to other objects outside its class. Making attributes private means that the attributes are only available to outside objects through the class methods, a technique called ***encapsulation***, or information hiding. A class diagram may show just the class name; or the class name and attributes; or the class name, attributes, and methods. Showing only the class name is useful when the diagram is very complex and includes many classes. If the

diagram is simpler, attributes and methods may be included. When attributes are included, there are three ways to show the attribute information. The simplest is to include only the attribute name, which takes the least amount of space. The type of data (such as string, double, or date) may be included on the class diagram. The most complete descriptions would include an equal sign (=) after the type of data followed by the initial value for the attribute.

Figure 5.3 illustrates class attributes. If the attribute must take on one of a finite number of values, such as a student type with values of F for full-time, P for part-time,and N for nonmatriculating, these may be included in curly brackets separated by commas: studentType:char{F,P,N}.

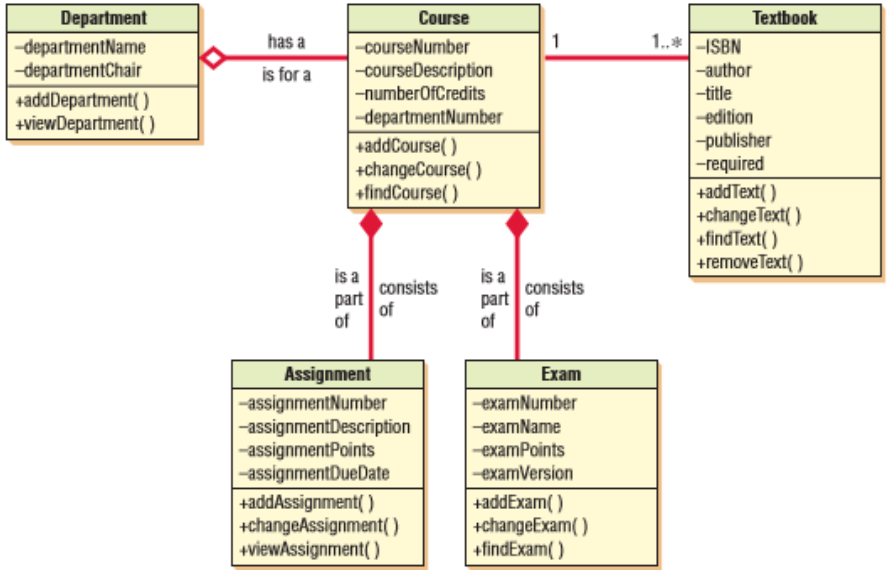


Figure 5.3: A class diagram for course offerings. The filled-in diamonds show aggregation and the empty diamond shows a whole-part relationship.

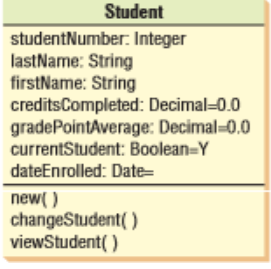


Figure 5.4: An extended Student class.

Information hiding means that objects' methods must be available to other classes, so methods are often public, meaning that they may be invoked from other classes. On a class diagram, public messages (and any public attributes) are shown with a plus sign (+) in front of them. Methods also have parentheses after them, indicating that data may be passed as parameters along with the message. The message parameters, as well as the type of data, may be included on the class diagram. There are two types of methods:standard and custom. ***Standard methods*** are basic things that all classes of objects know how to do, such as create a new object instance. ***Custom methods*** are designed for a specific class.

5.2.1 Inheritance

Another key concept of object-oriented systems is inheritance. Classes can have children; that is, one class can be created out of another class. In UML, the original—or parent—class is known as a base class. The child class is called a derived class. A ***derived class*** can be created in such a way that it will inherit all the attributes and behaviors of the base class. A derived class, however, may have additional attributes and behaviors. Inheritance reduces programming labor by using common objects easily.

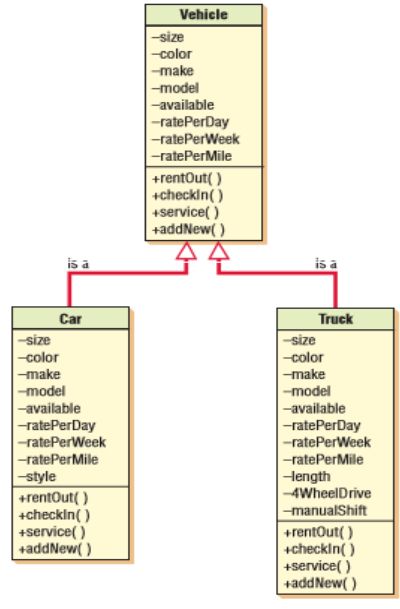


Figure 5.5: A class diagram showing inheritance.

Car and Truck are specific examples of vehicles and inherit the characteristics of the more general class, Vehicle.

5.2.2 Method Overloading

Method overloading refers to including the same method (or operation) several times in a class. The ***method signature*** includes the method name and the parameters included with the method. The same method may be defined more than once in a given class, as long as the parameters sent as part of the message are different; that is, there must be a different message signature. There may be a different number of parameters, or the parameters might be a different type, such as a number in one method and a string in another method. An example of method overloading may be found in the use of a plus sign in many programming languages. If the attributes on either side of the plus sign are numbers, the two numbers are added. If the attributes are strings of characters, the strings are concatenated to form one long string.

5.2.3 Relationships

Another way to enhance class diagrams is to show relationships. These are shown as lines connecting classes on a class diagram. The following figure, is an entity-relationship (E-R) diagram that shows various types of relationships.

There are two categories of relationships: associations and whole/part relationships.

ASSOCIATIONS. The simplest type of relationship is an association, or a structural connection between classes or objects. Associations are shown as a simple line on a class diagram. The end points of the line are labeled with a symbol indicating the ***multiplicity***, which is the same as ***cardinality*** on an entity-relationship diagram. The notation 0..1 represents from zero to one, and the notation 1..* represents from one to many. Class diagrams do not restrict the lower limit for an association. For example, an association might be 5..*, indicating that a minimum of five must be present. The same is true for upper limits. For example, the number of courses a student is currently enrolled in may be 1..10, representing from 1 to 10 courses. It can also include a range of values separated by commas, such as 2, 3, 4. Associations are illustrated in Figure 5.6.

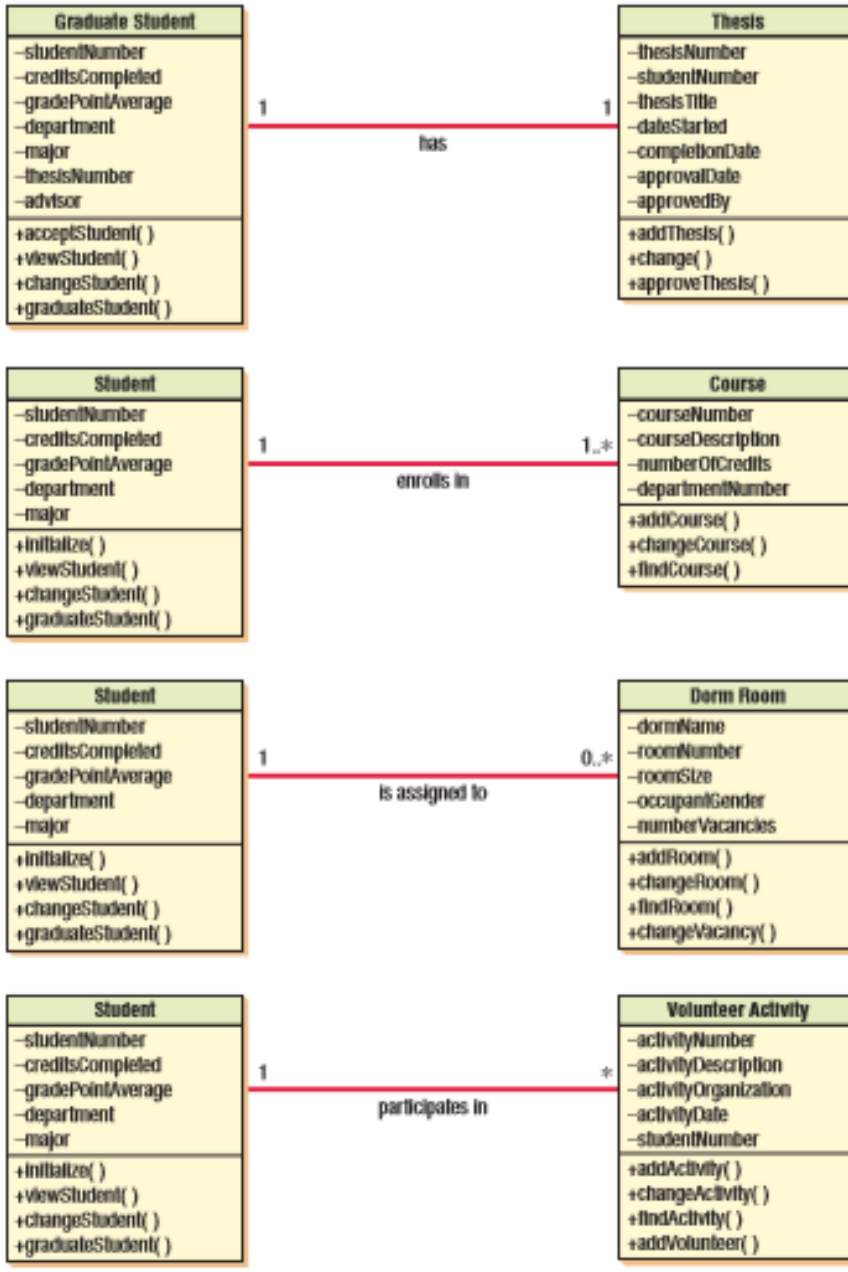


Figure 5.6: Types of associations that may occur in class diagrams.

In the UML model, associations are usually labeled with a descriptive name. Association classes are those that are used to break up a many-to-many association between classes. These are similar to associative entities on an entity-relationship diagram. Student and Course have a many-to-many relationship, which is resolved by adding an association class called Section between the classes of Student and Course. Figure 5.7 illustrates an association class called Section, shown with a dotted line connected to the many-to-many relationship line.

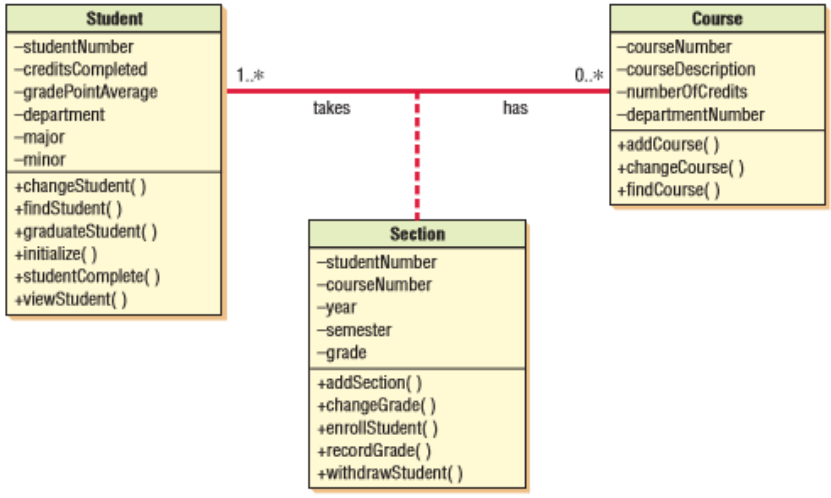


Figure 5.7: An example of an associative class.

An object in a class may have a relationship to other objects in the same class, called a ***reflexive association***. An example would be a task having a precedent task, or an employee supervising another employee. This is shown as an association line connecting the class to itself, with labels indicating the role names, such as task and precedent task.

WHOLE/PART RELATIONSHIPS. Whole/part relationships are when one class represents the whole object and other classes represent parts. The whole acts as a container for the parts. These relationships are shown on a class diagram by a line with a diamond on one end. The diamond is connected to the object that is the whole. Whole/part relationships (as well as aggregation) are shown in Figure 5.8. A whole/part relationship may be an entity object that has distinct parts, such as a computer system that includes the computer, printer, display, and so on, or an automobile that has an engine, brake system,

transmission, and so on. Whole/part relationships may also be used to describe a user interface, in which one GUI screen contains a series of objects such as lists, boxes, or radio buttons, or perhaps a header, body, and footer area. Whole/part relationships have three categories: aggregation, collection, and composition.

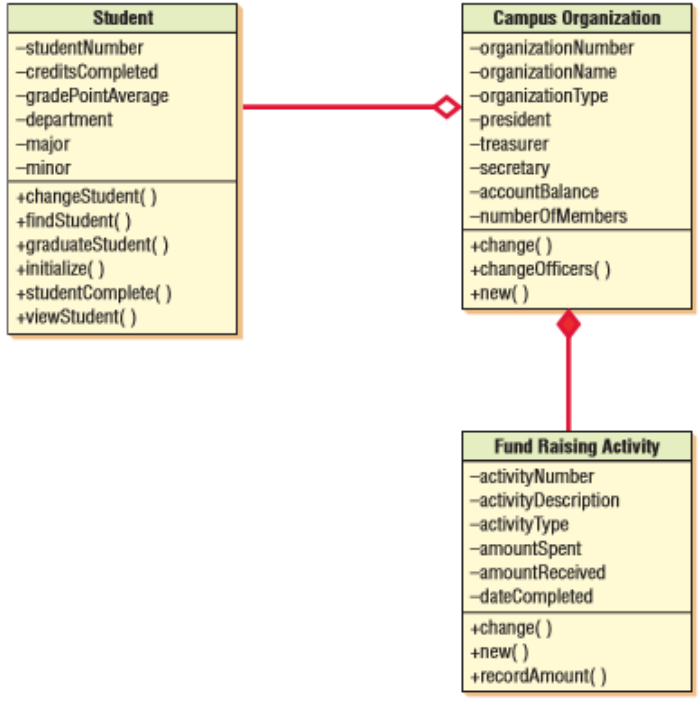


Figure 5.8: An example of whole/part and aggregation relationships.

An ***aggregation*** is often described as a "has a" relationship. Aggregation provides a means of showing that the whole object is composed of the sum of its parts (other objects). In the student enrollment example, the department has a course and the course is for a department. This is a weaker relationship, because a department may be changed or removed and the course may still exist. A computer package may not be available any longer, but the printers and other components still exist. The diamond at the end of the relationship line is not filled in.

A ***collection*** consists of a whole and its members. This may be a voting district with voters or a library with books. The voters or books may change, but the whole retains its identity. This is a weak association.

Composition, a whole/part relationship in which the whole has a responsibility for the part, is a stronger relationship, and is usually shown with a filled-in diamond. Keywords for composition are one class "always contains" another class. If the whole is deleted, all parts are deleted. An example would be an insurance policy with riders. If the policy is canceled, the insurance riders are also canceled. In a database, the referential integrity would be set to delete cascading child records. In a university there is a composition relationship between a course and an assignment as well as between a course and an exam. If the course is deleted,assignments and exams are deleted as well.

5.2.4 Generalization/Specialization (Gen/Spec) Diagrams

A generalization/specialization (gen/spec) diagram may be considered to be an enhanced class diagram. Sometimes it is necessary to separate out the generalizations from the specific instances. As we mentioned at the beginning of this chapter, a koala bear is part of a class of marsupials, which is part of a class of animals. Sometimes we need to distinguish whether a koala bear is an animal or a koala bear is a type of animal. Furthermore, a koala bear can be a stuffed toy animal. So we often need to clarify these subtleties.

GENERALIZATION. A generalization describes a relationship between a general kind of thing and a more specific kind of thing. This type of relationship is often described as an "is a"relationship. For example, a car is a vehicle and a truck is a vehicle. In this case, vehicle is the general thing, whereas car and truck are the more specific things. Generalization relationships are used for modeling class inheritance and specialization. A ***general class*** is sometimes called a superclass, base class, or parent class; a ***specialized class*** is called a subclass, derived class, or child class.

INHERITANCE. Several classes may have the same attributes and/or methods. When this occurs, a general class is created containing the common attributes and methods. The specialized class inherits or receives the attributes and methods of the general class. In addition, the specialized class has attributes and methods that are unique and only defined in the specialized class. Creating generalized classes and allowing the specialized class to inherit the attributes and methods helps to foster reuse, because the code is used many times. It also helps to maintain existing program code. This allows the analyst

to define attributes and methods once but use them many times, in each inherited class.

POLYMORPHISM. (meaning many forms), or method overriding (not the same as method overloading), is the capability of an object-oriented program to have several versions of the same method with the same name within a superclass/subclass relationship. The subclass inherits a parent method but may add to it or modify it. The subclass may change the type of data, or change how the method works. For example, there might be a customer who receives an additional volume discount, and the method for calculating an order total is modified. The subclass method is said to override the superclass method. When attributes or methods are defined more than once, the most specific one (the lowest in the class hierarchy) is used. The compiled program walks up the chain of classes, looking for methods.

ABSTRACT CLASSES. are general classes and are used when gen/spec is included in the design. The general class becomes the abstract class. The abstract class has no direct objects or class instances, and is only used in conjunction with specialized classes. Abstract classes usually have attributes and may have a few methods. Figure 5.9 is an example of a gen/spec class diagram. The arrow points to the general class, or superclass. Often the lines connecting two or more subclasses to a superclass are joined using one arrow pointing to the superclass, but these could be shown as separate arrows as well. Notice that the top level is Person, representing any person. The attributes describe qualities that all people at a university have. The methods allow the class to change the name and the address (including telephone and email address). This is an abstract class, with no instances. Student and Employee are subclasses, because they have different attributes and methods. An employee does not have a grade point average and a student does not have a salary. This is a simple version, and does not include employees that are students and students that work for the university. If these were added, they would be subclasses of the Employee and Student classes. Employee has two subclasses, Faculty and Administrator, because there are different attributes and methods for each of these specialized classes. Subclasses have special verbs to define them. These are often run-on words, using isa for “is a,” isakinda for “is a kind of,” and canbea for “can be a.” There is no distinction between “is a” and “is an;” they both use isa.

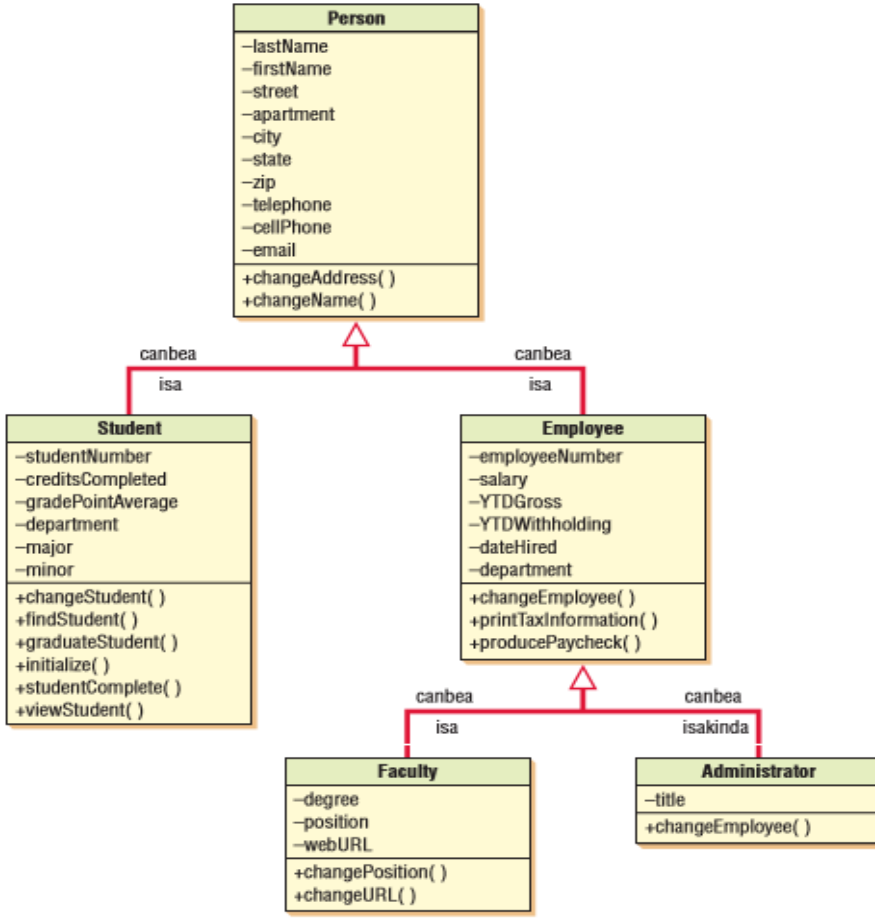


Figure 5.9: A gen/spec diagram is a refined form of a class diagram.

The following diagram refers to the company conceptual schema in UML class diagram notation.

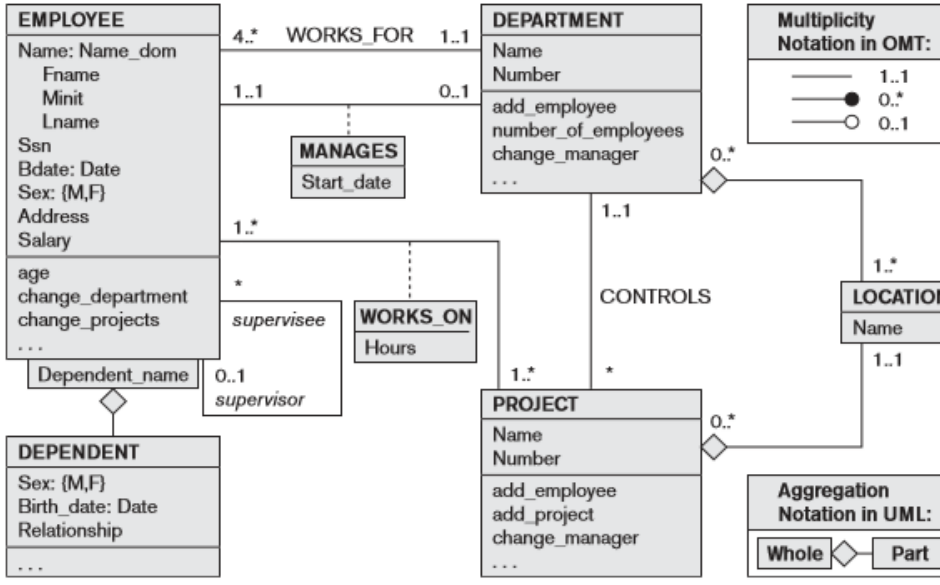


Figure 5.10: the company conceptual schema.

5.3 Sequence and Communication Diagrams

An interaction diagram is either a sequence diagram or a communication diagram, both of which show essentially the same information. These diagrams, along with class diagrams, are used in a use case realization, which is a way to achieve or accomplish a use case.

5.3.1 Sequence Diagrams

Sequence diagrams can illustrate a succession of interactions between classes or object instances over time. Sequence diagrams are often used to illustrate the processing described in use case scenarios. In practice, sequence diagrams are derived from use case analysis and are used in systems design to derive the interactions, relationships, and methods of the objects in the system.

Sequence diagrams are used to show the overall pattern of the activities or interactions in a use case. Each use case scenario may create one sequence diagram, although sequence diagrams are not always created for minor scenarios. The symbols used in sequence diagrams are shown in Figure 27.

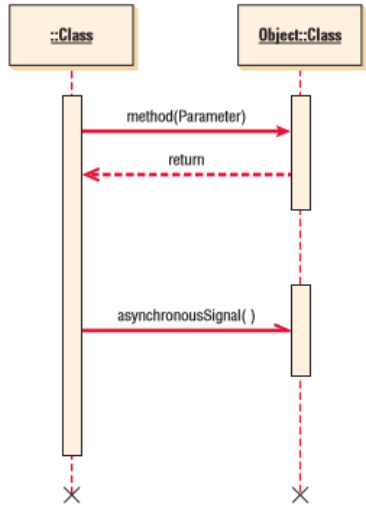


Figure 5.11: Specialized symbols used to draw a sequence diagram.

Actors and classes or object instances are shown in boxes along the top of the diagram. The left most object is the starting object and may be a person (for which a use case actor symbol is used), window, dialog box, or other user interface. Some of the interactions are physical only, such as signing a contract. The top rectangles use indicators in the name to indicate whether the rectangle represents an object, a class, or a class and object.

objectName:	**A name with a colon after it represents an object.**
:class	**A colon with a name after it represents a class.**
objectName:class	**A name, followed by a colon and another name, represents an object in a class.**

A vertical line represents the lifeline for the class or object, which corresponds to the time from when it is created through when it is destroyed. An X on the bottom of the lifeline represents when the object is destroyed. A lateral bar or vertical rectangle on the lifeline shows the focus of control when the object is busy doing things. Horizontal arrows show messages or signals that are sent between the classes. Messages belong to the receiving class. There are some variations in the message arrows. Solid arrowheads represent synchronous calls, which are the most common. These are used when the sending class waits for a response from the receiving class, and control is returned to the sending class when the class receiving the message finishes

executing. Half (or open) arrowheads represent asynchronous calls, or those that are sent without an expectation of returning to the sending class. An example would be using a menu to run a program. A return is shown as an arrow, sometimes with a dashed line. Messages are labeled using one of the following formats:

- The name of the message followed by empty parentheses: messageName().
- The name of the message followed by parameters in parentheses: messageName(parameter1, parameter2 . . .).
- The message name followed by the parameter type, parameter name, and any default value for the parameter in parentheses: messageName(parameterType:parameterName(defaultValue). Parameter types indicate the type of data, such as string, number, or date.
- The message may be a stereotype, such as «Create», indicating that a new object is created as a result of the message.

Timing in the sequence diagram is displayed from top to bottom; the first interaction is drawn at the top of the diagram, and the interaction that occurs last is drawn at the bottom of the diagram. The interaction arrows begin at the bar of the actor or object that initiates the interaction, and they end pointing at the bar of the actor or object that receives the interaction request. The starting actor, class, or object is shown on the left. This may be the actor that initiates the activity or it may be a class representing the user interface.

Sequence diagrams can be used to translate the use case scenario into a visual tool for systems analysis. The initial sequence diagram used in systems analysis shows the actors and classes in the system and the interactions between them for a specific process. You can use this version of the sequence diagram to verify processes with the business area experts who have assisted you in developing the system requirements. A sequence diagram emphasizes the time ordering (sequence) of messages.

During the systems design phase, the sequence diagrams are refined to derive the methods and interactions between classes. Messages from one class are used to identify class relationships. The actors in the earlier sequence diagrams are translated to interfaces, and class interactions are translated to class methods. Class methods used to create instances of other classes and to perform other internal system functions become apparent in the system design using sequence diagrams.

5.3.2 Communication Diagrams

Communication diagrams were introduced in UML 2.0. Their original name in UML 1.x was collaboration diagrams. Communication diagrams describe the interactions of two or more things in the system that perform a behavior that is more than any one of the things can do alone. For instance, a car can be broken down into several thousand individual parts. The parts are put together to form the major subsystems of the vehicle: the engine, the transmission, the brake system, and so forth. The individual parts of the car can be thought of as classes, because they have distinct attributes and functions. The individual parts of the engine form a collaboration, because they "communicate" with each other to make the engine run when the driver steps on the accelerator.

A communication diagram is made up of three parts: objects (also called participants), the communication links, and the messages that can be passed along those links. Communication diagrams show the same information as a sequence diagram but may be more difficult to read. In order to show time ordering, you must indicate a sequence number and describe the message. A *communication diagram* emphasizes the organization of objects, whereas a *sequence diagram* emphasizes the time ordering of messages. A communication diagram will show a path to indicate how one object is linked to another. A communication diagram for the student admission example is illustrated in Figure 5.12. Each rectangle represents an object or a class. Connecting lines show the classes that need to collaborate or work with each other. The messages sent from one class to another are shown along connecting lines. Messages are numbered to show the time sequence. Return values may also be included and numbered to indicate when they are returned within the time sequence.

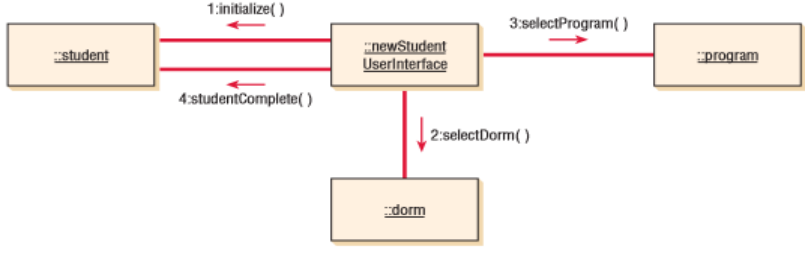


Figure 5.12: A communication diagram for student admission.

Communication diagrams show the same information that is depicted in a sequence diagram but emphasize the organization of objects rather than the time ordering.

5.4 Activity Diagram

Activity diagrams show the sequence of activities in a process, including sequential and parallel activities, and decisions that are made. An activity diagram is usually created for one use case and may show the different possible scenarios. The symbols on an activity diagram are illustrated in Figure 5.13.

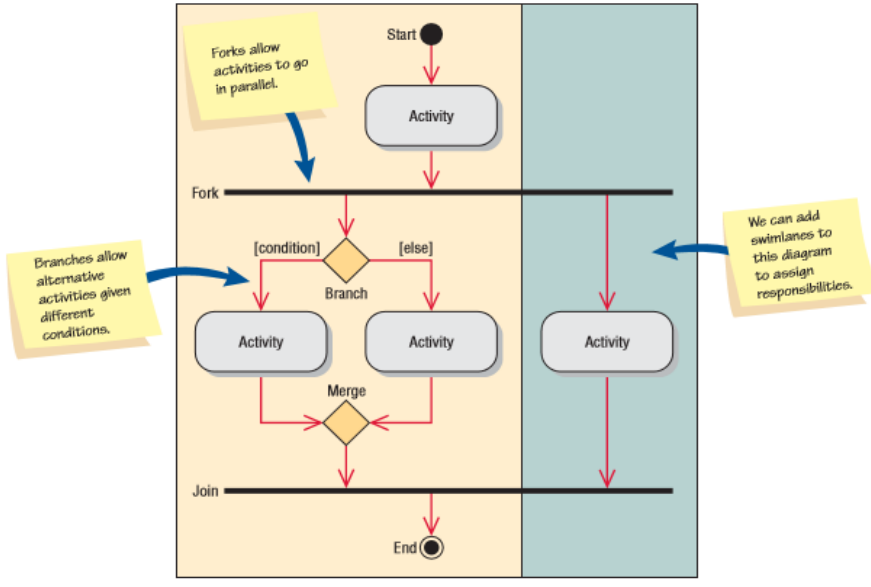


Figure 5.13: Specialized symbols are used to draw an activity diagram.

A rectangle with rounded ends represents an ***activity***, either a manual one, such as signing a legal document, or an automated one, such as a method or program. An arrow represents an event. ***Events*** represent things that happen at a certain time and place. A diamond represents either a decision (also called a branch) or a merge. ***Decisions*** have one arrow going into the diamond and several going out. A guard condition, showing the ***condition values***, may be included. Merges show several events combining to form one event. A long, flat rectangle represents a synchronization bar. These are used to show parallel activities, and may have one event going into the synchronization bar and

several events going out of it, called a ***fork***. A synchronization in which several events merge into one event is called a ***join***.

There are two symbols that show the start and end of the diagram. The initial state is shown as a filled-in circle. The final state is shown as a black circle surrounded by a white circle. Rectangles surrounding other symbols, called swimlanes, indicate partitioning and are used to show which activities are done on which platform, such as a browser, server, or mainframe computer; or to show activities done by different user groups. Swimlanes are zones that can depict logic as well as the responsibility of a class. Activity diagrams are not used for all use cases. Use the activity diagram when:

1. It helps to understand the activities of a use case.
2. The flow of control is complex.
3. There is a need to model workflow.
4. All scenarios need to be shown.

The analyst would not need an activity diagram when the use case is simple or there is a need to model the change of state. Activity diagrams may also be used to model a lower-level method, showing detailed logic.

5.5 StateChart Diagram

The statechart, or state transition, diagram is another way to determine class methods. It is used to examine the different states that an object may have. A statechart diagram is created for a single class. Typically objects are created, go through changes, and are deleted or removed. Objects exist in these various states, which are the conditions of an object at a specific time. An object's attribute values define the state that the object is in, and sometimes there is an attribute, such as Order Status (pending, picking, packaged, shipped, received, and so on) that indicates the state. A ***state*** has a name with each word capitalized. The name should be unique and meaningful to the users. A state also has entry and exit actions, the things the object must do every time it enters or leaves a given state. An ***event*** is something that happens at a specific time and place. Events cause a change of the object state, and it is said that a transition "fires." States separate events, such as an order that is waiting to be filled, and events separate states, such as an Order Received event or an Order Complete event. An event causes the transition, and happens when a guard condition has been met. A ***guard condition*** is something that evaluates to either true or false, and may be as simple as "Click to confirm

order." It also may be a condition that occurs in a method, such as an item that is out of stock. Guard conditions are shown in square brackets next to the event label.

Each time an object changes state, some of the attributes change their values. Furthermore, each time an object's attributes change, there must be a method to change the attributes. Each of the methods would need a display or Web form to add or change the attributes.These become the interface objects. The display or Web form would often have more controls (or fields) on them than just the attributes that change. They would usually have primary keys, identifying information (such as a name or address), and other attributes that are needed for a good user interface.The exception is a temporal event, which may use database tables or a queue containing the information. A statechart diagram showing how a student progresses from a potential student to a graduated student is illustrated in Figure 5.14.

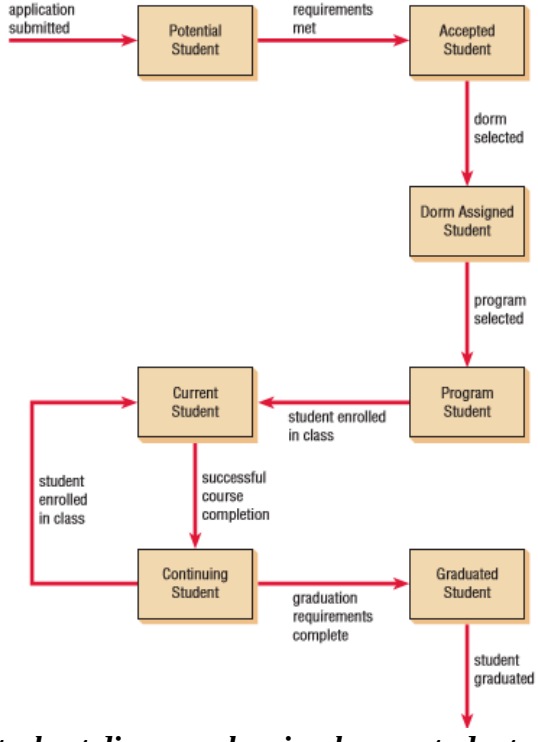


Figure 5.14: A statechart diagram showing how a student progresses from a potential student to a graduated student.

States are represented by rectangles, and events or activities are the arrows that link the states and cause one state to change to another state. Transition events are named in the past tense, because they have already occurred to create the transition. Statechart diagrams are not created for all classes. They are created when:

1. A class has a complex life cycle.
2. An instance of a class may update its attributes in a number of ways through the life cycle.
3. A class has an operational life cycle.
4. Two classes depend on each other.
5. The object's current behavior depends on what happened previously.

When you examine a statechart diagram, use the opportunity to look for errors and exceptions. Inspect the diagram to see whether events are happening at the wrong time. Also check that all events and states have been represented. Statechart diagrams have only two problems to avoid. Check to see that a state does not have all transitions going into the state or all transitions coming out of the state.

Each state should have at least one transition in and out of it. Some statechart diagrams use the same start and terminator symbols that an activity diagram uses: a filled-in circle to represent the start, and concentric circles with the center filled in to signify the end of the diagram.

5.6 Package Diagram and other UML artifacts

Packages are containers for other UML things, such as use cases or classes. Packages can show system partitioning, indicating which classes or use cases are grouped into a subsystem, called ***logical packages***. They may also be component packages, which contain physical system components, or use case packages,containing a group of use cases. Packages use a folder symbol with the package name either in the folder tab or centered in the folder. Packaging can occur during systems analysis, or later when the system is being designed. Packages may also have relationships, similar to class diagrams, which may include associations and inheritance.

As you continue constructing diagrams, you will want to make use of component diagrams, deployment diagrams, and annotational things. These permit different perspectives on the work being accomplished. The component diagram is similar to a class diagram, but is more of a bird's-eye view of the

system architecture. The component diagram shows the components of the system,such as a class file, a package, shared libraries, a database, and so on, and how they are related to each other. The individual components in a component diagram are considered in more detail within other UML diagrams, such as class diagrams and use case diagrams.

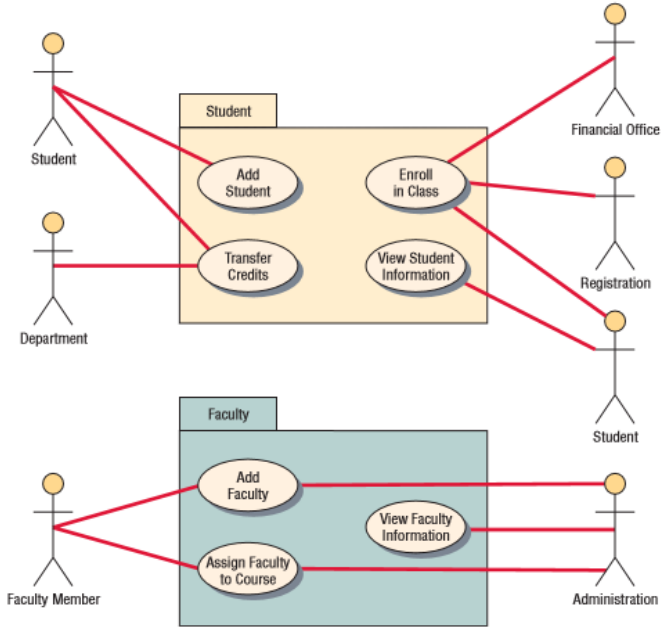


Figure 5.15: Use cases can be grouped into packages.

The deployment diagram illustrates the physical implementation of the system,including the hardware, the relationships between the hardware, and the system on which it is deployed. The deployment diagram may show servers, workstations, printers, and so on. Annotational things give developers more information about the system. These consist of notes that can be attached to anything in UML: objects, behaviors, relationships, diagrams, or anything that requires detailed descriptions, assumptions, or any information relevant to the design and functionality of the system. The success of UML relies on the complete and accurate documentation of your system model to provide as much information as possible to the development team. Notes provide a source of common knowledge and understanding about your system to help put your developers on the same page. Notes are shown as a paper symbol with a bent corner and a line connecting them to the area that needs elaboration.

References

[1] Management Information Systems: Managing The Digital Firm, Laudon, KC., And Laudon, JP, 12th Edition, 2012.

[2] Management Information Systems, O'Brien, JA and Marakas, GM, 10th Edition, 2007.

[3] Introduction To Information Systems, O'Brien, JA and Marakas, GM, 10th Edition, 16th Edition, 2011.

[4] Fundamentals of Database System, R. Elmasri, 7th Edition, 2016.

[5] Systems Analysis And Design, KENDALL, KE., and KENDALL, JE., 8th Edition, 2006.