EVELYN HONE COLLEGE OF APPLIED ARTS AND COMMERCE

BUSINESS STUDIES DEPARTMENT

COMPUTER SECTON

SYSTEMS ANALYIS AND DESIGN II

DESIGN OF SYSTEMS OUTPUT

One of the most important features of an Information System for users is the output it produces. Without quality output, the entire system may appear to be so unnecessary that users will avoid using it, possibly causing it to fail.

Output Design Objectives

Because useful output is essential to ensuring the use and acceptance of the information system, there are six objectives that the systems analyst tries to attain when designing output:

1. Designing output to serve the intended purpose.
2. Designing output to fit the user.
3. Delivering the appropriate quantity of output.
4. Making sure the output is where it is needed.
5. Providing the output on time.
6. Choosing the right output method.

Designing Output to Serve the Intended Purpose

All output should have a purpose. During the information requirements determination phase of analysis, the systems analyst finds out what user and organizational purposes exist. Output is then designed based on those purposes. You will have numerous opportunities to supply output simply because the application permits you to do so. Remember the rule of purposiveness, however. If the output is not functional, it should not be created, because there are costs of time and materials associated with all output from the system.

Designing Output to Fit the User

With a large information system serving many users for many different purposes, it is often difficult to personalize output. On the basis of interviews, observations, cost considerations, and perhaps prototypes, it will be possible to design output that addresses what many, if not all, users need and prefer. Generally speaking, it is more practical to create user-specific or user-customizable output when designing for a decision support system or other highly interactive applications such as those using the Web as a platform. It is still possible, however, to design output to fit a user's tasks and function in the organization, which leads us to the next objective.

Delivering the Appropriate Quantity of Output

Part of the task of designing output is deciding what quantity of output is correct for users. A useful heuristic is that the system must provide what each person needs to complete his or her work. This answer is still far from a total solution, because it may be appropriate to display a subset of that information at first and then provide a way for the user to access additional information easily. The problem of information overload is so prevalent that it is a cliché, but it remains a valid concern. No one is served if excess information is given only to flaunt the capabilities of the system. Always keep the decision makers in mind. Often they will not need great amounts of output, especially if there is an easy way to access more via a hyperlink or drill-down capability.

Making Sure the Output Is Where It Is Needed

Output is often produced at one location and then distributed to the user. The increase in online, screen-displayed output that is personally accessible has cut down somewhat on the problem of distribution, but appropriate distribution is still an important objective for the systems analyst. To be used and useful, output must be presented to the right user. No matter how well designed reports are, if they are not seen by the pertinent decision makers, they have no value.

Providing the Output on Time

One of the most common complaints of users is that they do not receive information in time to make necessary decisions. Although timing isn't everything, it does play a large part in how useful output will be. Many reports are required on a daily basis, some only monthly, others annually, and others only by exception. Using well-publicized, Web-based output can alleviate some problems with the timing of output distribution as well. Accurate timing of output can be critical to business operations.

Choosing the Right Output Method

Choosing the right output method for each user is another objective in designing output. Much output now appears on display screens, and users have the option of printing it out with their own printer. The analyst needs to recognize the trade-offs involved in choosing an output method. Costs differ; for the user, there are also differences in the accessibility, flexibility, durability, distribution, storage and retrieval possibilities, transportability, and overall impact of the data. The choice of output methods is not trivial, nor is it usually a foregone conclusion.

How to identify computer output needs

Designing computer output should proceed in an organized, well thought out manner: the right output must be developed while ensuring that each output element is designed so that people will find the system easy to use effectively. The term output applies to any information produced by an Information System, whether printed or display. When analysts design computer output, they:

- Identify the specific output that is needed to meet the information requirements.
- Select methods for presenting information.
- Create document, report, or other formats that contain information produced by the system.

The methods of output vary across systems. For some, such as an inventory report of the quantity of merchandise, the computer system, under program control, simply retrieves on the hand from storage(usually in secondary storage) and assembles them into a presentable form. Little if any calculation is needed, since the data already exist and therefore need only to be retrieved. Other output may require substantial processing before the data are available for use. For example, to produce output on the cost to manufacture a product, the system first locates descriptions of all the parts used in the final product and the necessary quantity of each one used in the final product.

The output from an information system should accomplish one or more of the following objectives:

1) Convey information about past activities, current status, or projections of the future.
2) Signal important events, opportunities, problems, warnings
3) Trigger an action
4) Confirm an action.

Good systems output design cannot be developed without involving its end users. In other words, attractive output or even output that uses innovative computer technology cannot be classified as “good” unless it meets the needs of the organization and its users.

Key Output Questions

Answering the following questions fully and properly, helps systems analysts learn a great deal about what the output of a newly designed system should be:

1) Who will receive the output?

Is the user inside or outside the organization? External users may have specific requirements governing content, and media requirements that are unchangeable. Organizations may decide to package the same information differently when it is received by both internal and external users.

2) What is the planned use?

Does the output present information (quantity sales report), ask for a response (ask for a driver's license), or trigger an action (an overdue payment notice)? Usage determines content, form and media.

3) How much detail is needed?

Few items of data are needed to tell someone to renew a driver's license (requests for name, address, renewal date, and renewal fee, plus identification of the output as being a renewal notice). However, the quarterly report on sales contains many details varying formats that help get the message (what happened, how it happened, and what the outcome is) across to users. High and low quantities of data also suggest whether print or display methods should be used.

4) When and how often is the output needed?

Timing and timeliness guide specific designs. Some output will be produced infrequently and only when certain conditions arise: the license renewal notice every four years and payment notice when an account is overdue. However, the organization may require output every month for license renewals coming up in the next month or every week for customer accounts with new overdue balances for the week.

5) By what method?

Should the output be printed or displayed? The previous examples demonstrated output for which printed methods are used frequently. However, if a system responds to yes/no inquiries, a simple display response is often appropriate. Systems analysts should answer these questions for every output requirement.

The design process should not begin until you have answered those questions. To complete your understanding, you should meet with users to find out exactly what kind of output is needed. You can use prototypes and mock-ups to obtain feedback throughout the design process. Your answers will affect your output design strategies.

How to present information

In any system, how the information is presented will determine whether the output is clear and readable, the details convincing, and the decision making fast and accurate. Below are guidelines for presenting tabular and graphic information, the use of icons, and ways to utilize color.

a) Tabular Format

In general, end users are most accustomed to receiving information in tabular format. Accountants and those who regularly review financial data rely almost exclusively on tabular information (a row and column). Common examples of tabular reports are inventory control, accounts payable, ledger, sales analysts, and production scheduling reports. In general, the tabular format should be used under the following conditions:

- Details dominate and few narrative comments or explanations are needed.
- Details are presented in discrete categories
- Each category must be labeled.
- Totals must be drawn or comparisons made between components.

Certain information in tabular format is more important and should be more visible than other information. This will vary by application, but in general, we can be sure that the following items should stand out:

- Exceptions to the normal expectations
- Major categories or groups of activities or entities
- Summaries of major categories or activities
- Unique identification information
- Time dependent entities.

b) Graphic Format

Graphic systems are available across a wide range of prices and capabilities and for personal computers up to main frames. Management presentation has been enhanced by graphics and visuals for a long time. However, it is an area in computer information systems that is exploding, largely because low cost but powerful computer software is available that will use data in existing databases and produce high quality charts and diagrams. These capabilities are available on personal computers for a small investment, as well as on mainframes, where a full range of cost alternatives are available. While graphics can enhance information presentation, designing graphic output means more than just converting tabular details to graphic form.

Graphics are used for several reasons:

1) To improve the effectiveness of output reporting for targeted recipients
2) To manage information volume
3) To suit personal preferences.

c) Using Icons

Icons are pictorial representations of entities described by the data. For example, a picture of an automobile may be used in reporting new car sales over five different years. The level of sales for each year can be represented by the size of the icon for the automobile. Thus a doubling of sales between two years will be shown by making one icon twice as large as the other. Icons are now commonly used in computer interfaces to represent documents, file drawers and printers. They have been used by graphic artists for many years when reporting data in newspapers, annual reports to stakeholders and sales presentations. What advantages do icons offer over written descriptions or business graphics (such as pie chart and bar charts)? Properly selected icons communicate information immediately, since they duplicate images that users are already familiar with. Icons eliminate the necessity for users to learn abbreviations, notation, or special nomenclature. However, the appropriate icon ensures that the right words and phrases, and that the intended meaning is conveyed.

The following guidelines suggest when and how to use icons in system design:

- Select icons that will be immediately recognized and understood by the anticipated users, select from available standard icons.
- If there is no familiar icon for the situation, use narrative labels; avoid requiring users to learn and remember unfamiliar symbols or images.
- Use the same icon to represent identical concepts across multiple output elements.
- Avoid labeling icons; the image itself should communicate its meaning clearly without requiring a label.
- Use a layout that maintains space and avoids overcrowding between icons.
- Maintain a common size among different types of symbols (for example, cars versus trucks) unless showing performance differences (say, in sales).

Color presentation

Color facilities are also increasing in both large and small computer systems. However, as with graphics, improper use of color can hinder rather than help user and management productivity. Color should enhance, replace, good output design. In fact, a good practice for analysts is to first design the output in the best form possible. Only then should the use of color be considered.

In general, using four or fewer colors on a screen or report is recommended. The more colors used, the larger the data or shapes must be able to communicate information. If information is displayed visually, less can be shown clearly when four colors are used than when only two are included in the design. The analyst must take care to maintain consistent color usage throughout the output and reports for a system. For example, color red is excellent for highlighting exceptions and problems. Green and blue are best for representing normal situations. If an analyst specifies these color usages on one report in a system, they should be used consistently for all reporting in the system. The brightest colors emphasize the most important information on a display screen. Bright colors include white or pink. None of the colors will enhance or compensate for poor design, however, the effective use of graphics can improve the results managers and users can use when working with system output.

Types of Output

System output may be:

1) A **Report**: A report or account is any informational work (usually of writing, speech, television, or film) made with the specific intention of relaying information or recounting certain events in a widely presentable form. Written reports are documents which present focused, salient content to a specific audience. Reports are often used to display the result of a computation, an experiment, investigation, or inquiry etc. The audience may be public or private, an individual or the public in general. Reports are used in government, business, education, science, and other fields. Reports use features such as graphics, images, voice, or specialized vocabulary in order to persuade that specific audience to undertake an action.

2) A **Document**: A document is a writing that provides information especially information of an official nature. Documents are sometimes classified as secret, private or public. They may also be described as a draft or proof. When a document is copied, the source is referred to as the original. The page layout of a document is the manner in which information is graphically arranged in the document space (e.g., on a page). If the appearance of the document is of concern, page layout is generally the responsibility of a graphic designer.

2) A **Message**: A message in its most general meaning is an object of communication. It is a vessel which provides information yet, it can also be this information. Therefore, its meaning is dependent upon the context in which it is used; the term may apply to both the information and its form.

Output Technology

Although business information systems still provide most output as screen displays and printed matter, technology is having an enormous impact on how people communicate and obtain information. This trend is especially important to firms that use information technology to lower their costs, improve employee productivity, and communicate effectively with their customers.

In addition to screen output and printed matter, output can be delivered in the following ways:

Internet-Based Information Delivery

Millions of firms use the Internet to reach new customers and markets around the world. To support the explosive growth in e-commerce, Web designers must provide user-friendly screen interfaces that display output and accept input from customers. For example, a business can link its inventory system to its Web site so the output from the inventory system is displayed as an online catalog. Customers visiting the site can review the items, obtain current prices, and check product availability. Web-based delivery allows users to download a universe of files and documents to support their information needs. To reach prospective customers and investors, companies also use a live or prerecorded Webcast, which is can be an audio or video media file distributed over the Internet. Radio and TV stations also use this technique to broadcast program material to their audiences.

E-Mail

E-mail is an essential means of internal and external business communication. Employees send and receive e-mail on local or wide area networks, including the Internet. Companies send new product information to customers via e-mail, and financial services companies use e-mail messages to confirm online stock trades. Employees use e-mail to exchange documents, data, and schedules and to share business-related information they need to perform their jobs. In many firms, e-mail has virtually replaced traditional memos and printed correspondence.

Blogs

Web-based logs, called blogs, are another form of Web-based output. Because blogs are journals written from a particular point of view, they not only deliver facts to Web readers, but also provide opinions. Blogs are useful for posting news, reviewing current events, and promoting products.

Instant Messaging

This popular form of communication is another way for individuals and companies to communicate effectively over the Internet. Although some users feel that it can be a distraction, others like the constant flow of communication, especially as a team member in a collaborative situation.

Digital Audio, Images and Video

Sounds, images, and video clips can be captured, stored in digital format, and transmitted as output to users who can reproduce the content. Audio or video output can be attached to an e-mail message or inserted as a clip in a Microsoft Word document. Businesses also use automated systems to handle voice transactions and provide information to customers. For example, video clips provide online virtual tours that allow realtors to show off the best features of homes they are marketing. The user can zoom in or out, and rotate the image in any direction.

Podcasts

A podcast is a specially formatted digital audio file that can be downloaded by Internet users from a variety of content providers. Many firms use podcasts as sales and marketing tools, and to communicate with their own employees. Using software such as iTunes, you can receive a podcast, launch the file on your computer and store it on your portable player. Podcasts can include images, sounds, and video.

Automated Facsimile Systems

An automated facsimile or faxback system allows a customer to request a fax using e-mail, via the company Web site, or by telephone. The response is transmitted in a matter of seconds back to the user's fax machine. Although most users prefer to download documents from the Web, many organizations still offer an automated faxback service as another way to provide immediate response 24 hours a day.

Computer Output To Microfilm (Com)

Computer output to microfilm (COM) is often used by large firms to scan and store images of original documents to provide high-quality records management and archiving. COM systems are especially important for legal reasons, or where it is necessary to display a signature, date stamp, or other visual features of a document.

Computer Output to Digital Media

This process is used when many paper documents must be scanned, stored in digital format, and retrieved quickly. For example, if an insurance company stores thousands of paper application forms, special software can treat the documents as data and extract information from a particular column or area on the form. Digital storage media can include magnetic tape, CDs, DVDs, and high-density laser disks.

Specialized Forms of Output

An incredibly diverse marketplace requires many forms of specialized output. For example;

- ✓ Portable, Web-connected devices that can run multiple apps, handle multimedia output, and provide powerful, multipurpose communication for user
- ✓ Retail point-of-sale terminals that handle computer-based credit card transactions, print receipts, and update inventory records
- ✓ Automatic teller machines (ATMs) that can process bank transactions and print deposit and withdrawal slips
- ✓ Special-purpose printers that can produce labels, employee ID cards, driver's licenses, gasoline pump receipts, and, in some stares, lottery tickets
- ✓ Plotters that can produce high-quality images such as blueprints, maps, and electronic circuit diagrams
- ✓ Electronic detection of data embedded in credit cards, bank cards, and employee identification cards.

Printed and Screen Output

Although many organizations strive to reduce the flow of paper and printed reports, few firms have been able to eliminate printed output totally. Because they are portable, printed reports are convenient, and even necessary in some situations. Many users find it handy to view screen output, then print the information they need for a discussion or business meeting.

Printed output is also used in **turnaround documents**, which are output documents that are later entered back into the same or another information system. In some areas, your telephone or utility bill, for example, might be a turnaround document printed by the company’s billing system. When you return the required portion of the bill with your check, the bill is scanned into the company’s accounts receivable system to record the payment accurately.

Report Design

A *report* is a document that presents information in an organized format for a specific audience and purpose. Designers use a variety of styles, fonts, and images to produce reports that are attractive and user friendly. Whether printed or viewed on-screen, reports must be easy to read and well organized. Rightly or wrongly, some managers judge an entire project by the quality of the reports they receive.

Database programs such as Microsoft Access include a variety of report design tools, including a Report Wizard, which is a menu-driven feature that designers can use to create reports quickly and easily. Although the vast majority of reports are designed graphically, some systems still produce one or more character-based reports that use a character set with fixed spacing. Printing character-based reports on high-speed impact printers is a fast, inexpensive method for producing large-scale reports, such as payroll or inventory reports, or registration rosters at a school. This is especially true if multiple copies are required.

Types of Reports

To be useful, a report must include the information that a user needs. From a user's point of view, a report with too little information is of no value. Too much information, however, can make a report confusing and difficult to understand. When designing reports, the essential goal is to match the report to the user's specific information needs. Depending on their job functions, users might need one or more of the reports described below:

1. Detail Reports

A detail report produces one or more lines of output for each record processed. Because it contains one or more lines for each record, a detail report can be quite lengthy. Consider, for example, a large auto parts business. If the firm stocks 3,000 parts, then the detail report would include 3,000 detail lines on approximately 50 printed pages. A user who wants to locate any part in short supply has to examine 3,000 detail lines to find the critical items. A better alternative might be an exception report.

2. Exception Reports

An exception report displays only those records that meet a specific condition or conditions. Exception reports are useful when the user wants information only on records that might require action, but does not need to know the details. For example, a credit manager might use an exception report to identify only those customers with past-due accounts, or a customer service manager might want a report on all packages that were not delivered within a specified time period.

3. Summary Reports

Upper-level managers often want to see total figures and do not need supporting details. A sales manager, for example, might want to know total sales for each sales representative, but not want a detail report listing every sale made by them. In that case, a summary report is appropriate. Similarly, a personnel manager might need to know the total regular and overtime hours worked by employees in each store but might not be interested in the number of hours worked by each employee.

User Involvement

Users should approve all report designs in advance. The best approach is to prepare a sample report, called a mock-up, or prototype, for users to review. The sample should include typical field values and contain enough records to show all the design features. Depending on the type of printed output, you can create a Microsoft Word document or use a report generator to create mock-up reports.

Report Design Principles

Printed reports must be attractive, professional, and easy to read. For example, a well-designed report should provide totals and subtotals for numeric fields. Good report design requires effort and attention to detail. To produce a well-designed report, the analyst must consider design features such as report headers and footers, page headers and footers, column headings and alignment, column spacing, field order, and grouping of detail lines.

Report Headers and Footers

Every report should have a report header and a report footer. The report header, which appears at the beginning of the report, identifies the report, and contains the report title, date, and other necessary information. The report footer, which appears at the end of the report, can include grand totals for numeric fields and other end-of-report information.

Page Headers and Footers

Every page should include a page header, which appears at the top of the page and includes the column headings that identify the data. The headings should be short but descriptive. Avoid abbreviations unless you know that users will understand them clearly. Either a page header or a page footer, which appears at the bottom of the page, is used to display the report title and the page number.

Column Spacing

You should space columns of information carefully. A crowded report is hard to read, and large gaps between columns make it difficult for the eye to follow a line. Columns should stretch across the report, with uniform spacing and suitable margins at top, bottom, right, and left. Some report designers use landscape orientation when working with a large number of columns; others prefer to break the information into more than one report. In some cases, a smaller point size will solve the problem, but the report must remain readable and acceptable to users.

Field Order

Fields should be displayed and grouped in a logical order.

Consistent Design

Good report design includes standards to ensure that reports are uniform and consistent. When a system produces multiple reports, each report should share common design elements. For example, the date and page numbers should print in the same place on each report page. Abbreviations used in reports also should be consistent. Items in a report also should be consistent. If one report displays the inventory location as a shelf number column followed by a bin number column, that same layout should be used on all inventory location reports.

EVELYN HONE COLLEGE OF APPLIED ARTS AND COMMERCE

BUSINESS STUDIES DEPARTMENT

COMPUTER SECTON

SYSTEMS ANALYIS AND DESIGN II

DATA DESIGN

Data Design Concepts

Before constructing an information system, a systems analyst must understand basic data design concepts, including data structures and the characteristics of file processing and database systems, including Web-based database design.

Data Structures

A data structure is a framework for organizing and storing data in an information system.

Data structures consist of files or tables that are linked in various ways. Each file or table contains data about people, places, things, or events that interact with the information system. For example, one file or table might contain data about customers, and other files or tables might store data about products, orders, suppliers, or employees. Depending on how the system's files and tables are organized and linked, an information system is called either a file processing system or a database management system.

A file processing system, also called a **file-oriented system**, stores and manages data in one or more separate files. A major disadvantage of file-oriented systems is that the same data is stored in more than one place. A **database system** consists of linked tables that form one overall data structure. Compared to file processing, a database system offers much greater flexibility and efficiency. This link allows information to be accessed from either table as if the two tables were one large table, making it unnecessary to store duplicate information in both tables. File processing systems still exist to handle specific applications, but the vast majority of information systems today are designed as databases.

File Processing

In a typical file processing environment, a company might have three departments, each with its own information system and data files. Three potential problems exist in a file processing environment.

The first problem is data redundancy, which means that data common to two or more information systems is stored in several places. Data redundancy requires more storage space, and maintaining and updating data in several locations is expensive.

Second, data integrity problems can occur if updates are not applied in every file. Changing the data in only one of the systems will cause inconsistent data and result in incorrect information in the second system.

The third problem is the rigid data structure of a typical file processing environment. Businesses must make decisions based on company-wide data, and managers often require information from multiple business units and departments. In a file processing environment, that means retrieving information from independent, file-based systems, which is slow and inefficient.

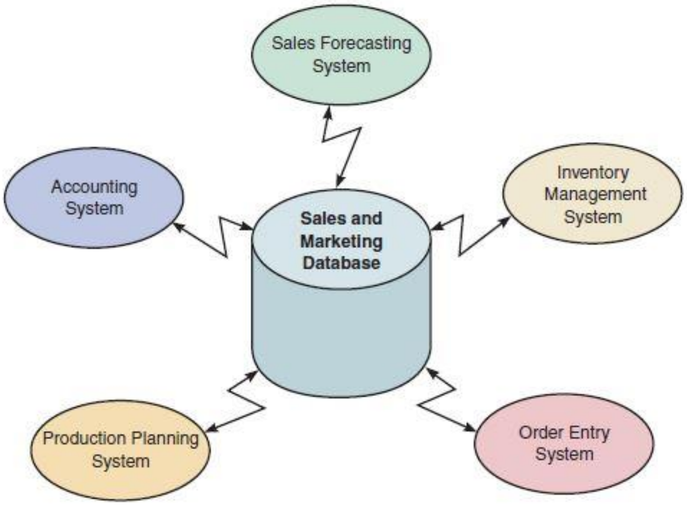
A file-oriented information system can contain various types of files, including master files, table files, transaction files, work files, security files, and history files.

- A master file stores relatively permanent data about an entity. For example, a PRODUCT master file contains one logical record for each product the company sells. The quantity field in each record might change daily, but other data, such as the product's code, name, and description, would not change.
- A table file contains reference data used by the information system. As with master files, table files are relatively static and are not updated by the information system. Examples of table files include tax tables and postage rate tables.
- A transaction file stores records that contain day-to-day business and operational data. A transaction file is an input file that updates a master file; after the update is completed, the transaction file has served its purpose. An example of a transaction file is a charges and payments file that updates a customer balance file.
- A work file is a temporary file created by an information system for a single task. Examples of work files include sorted files and report files that hold output reports until they are printed.
- A security file is created and saved for backup and recovery purposes. Examples of security files include audit trail files and backups of master, table, and transaction files. New security files must be created regularly to replace outdated files.

- A history file is a file created for archiving purposes. For example, students who have not registered for any course in the last two semesters might be deleted from the active student master file and added to an inactive student file, which is a history file that can be used for queries or reports.

Database Systems

A properly designed database system offers a solution to the problems of file processing. A database provides an overall framework that avoids data redundancy and supports a real-time, dynamic environment, two potential problems of a file processing system. In a file processing environment, data files are designed to fit individual business systems. In contrast, in a database environment, several systems can be built around a single database.


Sales Forecasting System
Accounting System
Sales and Marketing Database
Inventory Management System
Production Planning System
Order Entry System

A typical database environment might consist of a database serving five separate business systems.

Database Management System

A database management system (DBMS) is a collection of tools, features, and interfaces that enables users to add, update, manage, access, and analyze the contents of a set of data. From a user's point of view, the main advantage of a DBMS is that it offers timely, interactive, and flexible data access. Specific DBMS advantages include the following:

- Scalability, which means that a system can be expanded, modified, or downsized easily to meet the rapidly changing needs of a business enterprise. For example, if a company decides to add data about secondary suppliers of material it uses, a new table can be added to the relational database and linked with a common field.

- Better support for client/server systems. In a client/server system, processing is distributed throughout the organization. Client/server systems require the power and flexibility of a database design.
- Economy of scale. Database design allows better utilization of hardware. If a company maintains an enterprise-wide database, processing is less expensive using a powerful mainframe server instead of using several smaller computers. The inherent efficiency of high-volume processing on larger computers is called economy of scale.
- Flexible data sharing. Data can be shared across the enterprise, allowing more users to access more data. A database can be highly flexible, allowing users to view the same information in different ways. Users are empowered because they have access to the information they need to do their jobs.
- Enterprise-wide application. Typically, a DBMS is managed by a person called a database administrator (DBA), who assesses overall requirements and maintains the database for the benefit of the entire organization rather than a single department or user. Database systems can support enterprise-wide applications more effectively than file processing systems.
- Stronger standards. Effective database administration helps ensure that standards for data names, formats, and documentation are followed uniformly throughout the organization.
- Controlled redundancy. Redundancy means storing data in more than one place, which can result in inconsistency and data errors. Because the data is stored in a set of related tables, data items do not need to be duplicated in multiple locations. Even where some duplication is desirable for performance reasons, or disaster recovery, the database approach allows control of the redundancy.
- Better security. The DBA can define authorization procedures to ensure that only legitimate users can access the database and can allow different users to have different levels of access. Most DBMSs provide sophisticated security support.
- Increased programmer productivity. Programmers do not have to create the underlying file structure for a database. That allows them to concentrate on logical design and, therefore, a new database application can be developed more quickly than in a file-oriented system.
- Data independence. Systems that interact with a DBMS are relatively independent of how the physical data is maintained. That design provides the DBA flexibility to alter data structures without modifying information systems that use the data.

DBMS Components

A DBMS provides an interface between a database and users who need to access the data. Although users are concerned primarily with an easy-to-use interface and support for their business requirements, a systems analyst must understand all of the components of a DBMS. In addition to interfaces for users, database administrators, and related systems, a DBMS also has a data manipulation language, a schema and subschemas, and a physical data repository.

4. Users

Users typically work with predefined queries and switchboard commands, but also use query languages to access stored data. A **query language** allows a user to specify a task without specifying how the task will be accomplished. Some query languages use natural language commands that resemble ordinary English sentences. With a **query by example (QBE)** language, the user provides an example of the data requested. Many database programs also generate **SQL (Structured Query Language)**, which is a language that allows client workstations to communicate with servers and mainframe computers.

5. Database Administrators

A DBA is responsible for DBMS management and support. DBAs are concerned with data security and integrity, preventing unauthorized access, providing backup and recovery, audit trails, maintaining the database, and supporting user needs. Most DBMSs provide utility programs to assist the DBA in creating and updating data structures, collecting and reporting patterns of database usage, and detecting and reporting database irregularities.

6. Related Information Systems

A DBMS can support several related information systems that provide input to, and require specific data from, the DBMS. Unlike a user interface, no human intervention is required for two-way communication between the DBMS and the related systems.

7. Data Manipulation Language

A **data manipulation language (DML)** controls database operations, including storing, retrieving, updating, and deleting data. Most commercial DBMSs, such as Oracle and IBM’s DB/2, use a DML. Some database products, such as Microsoft Access, also provide an easy-to-use graphical environment that enables users to control operations with menu-driven commands.

8. Schema

The complete definition of a database, including descriptions of all fields, tables, and relationships, is called a **schema**. You also can define one or more subschemas. A **subschema** is a view of the database used by one or more systems or users. A subschema defines only those portions of the database that a particular system or user needs or is allowed to access. For example, to protect individual privacy, you might not want to allow a project management system to retrieve employee pay rates. In that case, the project management system subschema would not include the pay rate field. Database designers also use subschemas to restrict the level of access permitted. For example, specific users, systems, or locations might be permitted to create, retrieve, update, or delete data, depending on their needs and the company's security policies.

9. Physical Data Repository

A data dictionary describes all data elements included in the logical design. During data design, the data dictionary is transformed into a physical data repository, which also contains the schema and subschemas. The physical repository might be centralized, or it might be distributed at several locations. In addition, the stored data might be managed by a single DBMS, or several systems. To resolve potential database connectivity and access problems, companies use ODBC-compliant software that enables communication among various systems and DBMSs. ODBC, which stands for open database connectivity, is an industry-standard protocol that makes it possible for software from different vendors to interact and exchange data. ODBC uses SQL statements that the DBMS understands and can execute. Another common standard is called JDBC, or Java database connectivity. JDBC enables Java applications to exchange data with any database that uses SQL statements and is JDBC-compliant.

Data Design Terminology

Entity

An entity is a person, place, thing, or event for which data is collected and maintained.
For example, an online sales system may include entities named CUSTOMER,
ORDER, PRODUCT, and SUPPLIER.

Table or File

Data is organized into tables or files. A table, or file, contains a set of related records that store data about a specific entity. Tables and files are shown as two-dimensional structures that consist of vertical columns and horizontal rows. Each column represents a field, or characteristic of the entity, and each row represents a record, which is an individual instance, or occurrence of the entity. Although they can have different meanings in a specific context, the terms table and file often can be used interchangeably.

Field

A field, also called an attribute, is a single characteristic or fact about an entity. A common field is an attribute that appears in more than one entity. Common fields can be used to link entities in various types of relationships.

Record

A record, also called a tuple (rhymes with couple), is a set of related fields that describes one instance, or occurrence of an entity, such as one customer, one order, or one product. A record might have one or dozens of fields, depending on what information is needed.

Key Fields

During the systems design phase, you use key fields to organize, access, and maintain data structures. The four types of keys are primary keys, candidate keys, foreign keys, and secondary keys.

- **Primary Key** - A primary key is a field or combination of fields that uniquely and minimally identifies a particular member of an entity. For example, in a customer table the customer number is a unique primary key because no two customers can have the same customer number.
- **Combination key** – This is a primary key that is composed of two or more fields. A combination key is also called a composite key, a concatenated key, or a multi-valued key.
- **Candidate Key** - Sometimes you have a choice of fields or field combinations to use as the primary key. Any field that could serve as a primary key is called a candidate key. For example, if every employee has a unique employee number, then you could use either the employee number or the Social Security number as a primary key. Because you can designate only one field as a primary key, you should select the field that contains the least amount of data and is the easiest to use. Any field that is not a primary key or a candidate key is called a non-key field.
- **Foreign Key** - A foreign key is a field in one table that must match a primary key value in another table in order to establish the relationship between the two tables.
- **Secondary Key -** A secondary key is a field or combination of fields that can be used to access or retrieve records. Secondary key values are not unique. For example, if you need to access records for only those customers in a specific ZIP code, you would use the ZIP code field as a secondary key. Secondary keys also can be used to sort or display records in a certain order.

Entity-Relationship Diagrams

Recall that an entity is a person, place, thing, or event for which data is collected and maintained. For example, entities might be customers, sales regions, products, or orders. An information system must recognize the relationships among entities. For example, a customer entity can have several instances of an order entity.

An entity-relationship diagram (ERD) is a model that shows the logical relationships and interaction among system entities. An ERD provides an overall view of the system and a blueprint for creating the physical data structures.

Drawing an ERD

The first step is to list the entities that you identified during the systems analysis phase and to consider the nature of the relationships that link them. At this stage, you can use a simplified method to show the relationships between entities. Although there are different ways to draw ERDs, a popular method is to represent entities as rectangles and relationships as diamond shapes. The entity rectangles are labeled with singular nouns, and the relationship diamonds are labeled with verbs, usually in a top-to-bottom and left-to-right fashion. Unlike data flow diagrams, entity-relationship diagrams depict relationships, not data or information flows.

Types of Relationships

Three types of relationships can exist between entities: one-to-one, one-to-many, and many-to-many.

- A **one-to-one** relationship, abbreviated 1:1, exists when exactly one of the second entity occurs for each instance of the first entity. A number 1 is placed alongside each of the two connecting lines to indicate the 1:1 relationship.
- A **one-to-many** relationship, abbreviated 1: M, exists when one occurrence of the first entity can relate to many instances of the second entity, but each instance of the second entity can associate with only one instance of the first entity. For example, the relationship between DEPARTMENT and EMPLOYEE is one-to-many: One department can have many employees, but each employee works in only one department at a time.
- A **many-to-many** relationship, abbreviated M: N, exists when one instance of the first entity can relate to many instances of the second entity, and one instance of the second entity can relate to many instances of the first entity. The relationship between STUDENT and CLASS, for example, is many-to-many — one student can take many classes, and one class can have many students enrolled.

Cardinality

After an analyst draws an initial ERD, he or she must define the relationships in more detail by using a technique called cardinality. Cardinality describes the numeric relationship between two entities and shows how instances of one entity relate to instances of another entity. For example, consider the relationship between two entities: CUSTOMER and ORDER. One customer can have one order, many orders, or none, but each order must have one and only one customer. An analyst can model this interaction by adding cardinality notation, which uses special symbols to represent the relationship.

A common method of cardinality notation is called crow's foot notation because of the shapes, which include circles, bars, and symbols, that indicate various possibilities. A single bar indicates one, a double bar indicates one and only one, a circle indicates zero, and a crow's foot indicates many.

EVELYN HONE COLLEGE OF APPLIED ARTS AND COMMERCE

BUSINESS STUDIES DEPARTMENT

COMPUTER SECTON

SYSTEMS ANALYIS AND DESIGN II

SYSTMES IMPLEMENTATION

Managing Systems Implementation

Managing systems implementation involves application development, testing, documentation, training, data conversion, system changeover, and post-implementation evaluation of the results. During systems implementation, the system design specification serves as a blueprint for constructing the new system. The initial task is application development, which requires systems analysts and programmers to work together to construct the necessary programs and code modules. Before a changeover, the system must be tested and documented carefully, users must be trained, and existing data must be converted. After the new system is operational, a formal evaluation of the results takes place as part of a final report to management.

Software Quality Assurance

Quality Assurance is the review of system or software products and its documentation for assurance that the system meets the requirements and specifications.

- ✓ Purpose of QA is to provide confidence to the customers by constant delivery of product according to specification.
- ✓ Software quality Assurance (SQA) is a techniques that includes procedures and tools applied by the software professionals to ensure that software meet the specified standard for its intended use and performance.
- ✓ The main aim of SQA is to provide proper and accurate visibility of software project and its developed product to the administration.

- ✓ It reviews and audits the software product and its activities throughout the life cycle of system development.

Objectives of Quality Assurance

The objectives of conducting quality assurance are as follows:

- ✓ To monitor the software development process and the final software developed.
- ✓ To ensure whether the software project is implementing the standards and procedures set by the management.
- ✓ To notify groups and individuals about the SQA activities and results of these activities.
- ✓ To ensure that the issues, which are not solved within the software are addressed by the upper management.
- ✓ To identify deficiencies in the product, process, or the standards, and fix them.

Levels of Quality Assurance

There are several levels of QA and testing that need to be performed in order to certify a software product.

Level 1: Code Walk-through

At this level, offline software is examined or checked for any violations of the official coding rules. In general, the emphasis is placed on examination of the documentation and level of in-code comments.

Level 2: Compilation and Linking

At this level, it is checked that the software can compile and link all official platforms and operating systems.

Level 3: Routine Running

At this level, it is checked that the software can run properly under a variety of conditions such as certain number of events and small and large event sizes etc.

Level 4: Performance test

At this final level, it is checked that the performance of the software satisfies the previously specified performance level.

In today's competitive business environment, companies are intensely concerned with the quality of their products and services. A successful organization must improve quality in every area, including its information systems. Top management must provide the leadership, encouragement, and support needed for high-quality IT resources. No matter how carefully a system is designed and implemented, problems can occur. Rigorous testing can detect errors during implementation, but it is much less expensive to correct mistakes earlier in the development process. The main objective of quality assurance is to avoid problems or to identify them as soon as possible. Poor quality can result from inaccurate requirements, design problems, coding errors, faulty documentation, and ineffective testing. To improve the finished product, software systems developers should consider software engineering and internationally recognized quality standards.

Software Engineering

Software engineering is a software development process that stresses solid design, accurate documentation, and careful testing. The Software Engineering Institute (SEI) at Carnegie Mellon University is a leader in software engineering and provides quality standards and suggested procedures for software developers and systems analysts. SEI's primary objective is to find better, faster, and less-expensive methods of software development. To achieve that goal, SEI designed a set of software development standards called the Capability Maturity Model (CMM)'", which has been used successfully by thousands of organizations around the globe. The purpose of the model was to improve software quality, reduce development time, and cut costs. More recently, SEI established a new model, called Capability Maturity Model Integration (CMMI) that integrates software and systems development into a much larger framework called process improvement.

EVELYN HONE COLLEGE OF APPLIED ARTS AND COMMERCE

BUSINESS STUDIES DEPARTMENT

COMPUTER SECTON

SYSTEMS ANALYIS AND DESIGN II

SYSTMES IMPLEMENTATION

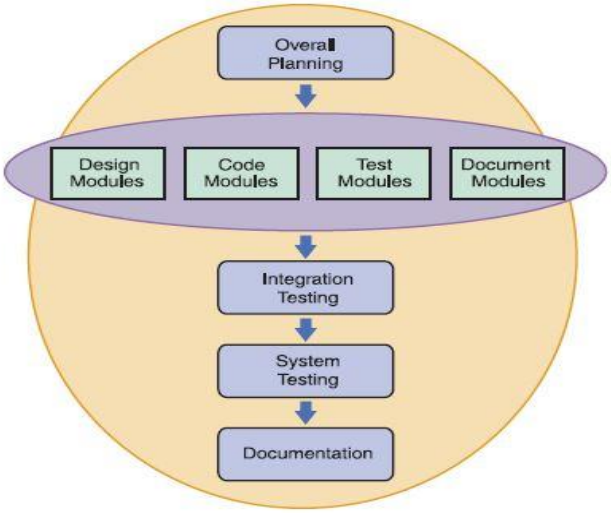
APPLICATION DEVELOPMENT

Application development is the process of constructing the programs and code modules that serve as the building blocks of the information system. Regardless of the method (structured approach, object oriented approach), the objective is to translate the design into program and code modules that will function properly. Because systems implementation usually is very labor-intensive, developers often use project management tools and techniques to control schedules and budgets.

Application Development Tasks

If you used traditional structured or object-oriented (O-O) methods, you now are ready to translate the design into a functioning application. Building a new system requires careful planning. After an overall strategy is established, individual modules must be designed, coded, tested, and documented. A module consists of related program code organized into small units that are easy to understand and maintain. After the modules are developed and tested individually, more testing takes place, along with thorough documentation of the entire system.

When you create program modules using structured or object-oriented methods, you start by reviewing documentation from prior SDLC phases and creating a set of program designs. If you built a documentation file early in the development process and updated it regularly, you now have a valuable repository of information. The centerpiece of your documentation is the system design specification, accompanied by diagrams, source documents, screen layouts, report designs, data dictionary entries, and user comments. If you used a CASE tool during the systems analysis and design process, your job will be much easier. At this point, coding and testing tasks begin. Although programmers typically perform the actual coding, IT managers usually assign systems analysts to work with them as a team.



The main steps in application development.

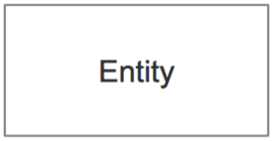
Systems Development Tools

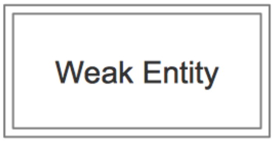
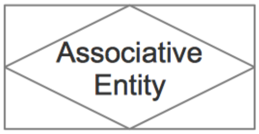
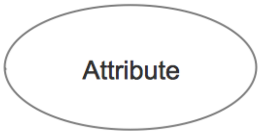
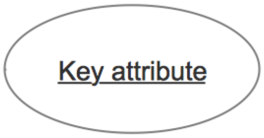
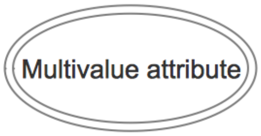
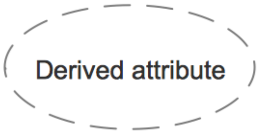
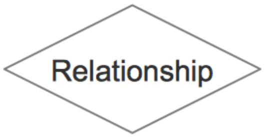
System developers use multipurpose tools to help them translate the system logic into properly functioning program modules. These generic tools include entity relationship diagrams, flowcharts, pseudo code, decision tables, and decision trees.

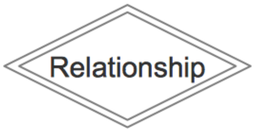
Entity-Relationship Diagrams

Entity-relationship diagrams are used to show the interaction among system entities and objects. An ERD is a useful tool regardless of which methodology you are using, because the various relationships (one-to-one, one-to-many, and many-to-many) must be understood and implemented in the application development process.

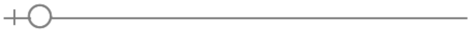
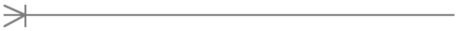
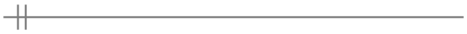
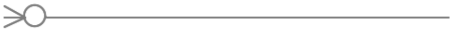
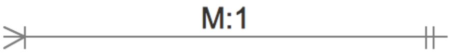
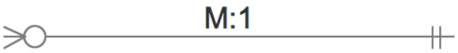
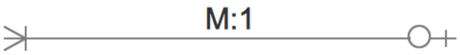
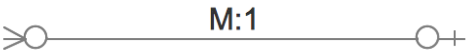
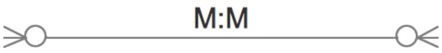
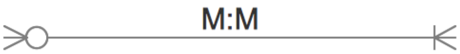
Entity Relationship Diagram Symbols — Chen notation

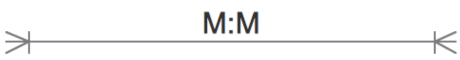
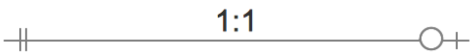
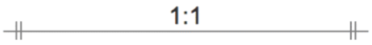
Symbol	Shape Name	Symbol Description
	Entities	
	Entity	An entity is represented by a rectangle which contains the entity's name.

	Weak Entity	An entity that cannot be uniquely identified by its attributes alone. The existence of a weak entity is dependent upon another entity called the owner entity. The weak entity's identifier is a combination of the identifier of the owner entity and the partial key of the weak entity.
	Associative Entity	An entity used in a many-to-many relationship (represents an extra table). All relationships for the associative entity should be many
Attributes		
	Attribute	In the Chen notation, each attribute is represented by an oval containing attribute's name
	Key attribute	An attribute that uniquely identifies a particular entity. The name of a key attribute is underscored.
	Multivalued attribute	An attribute that can have many values (there are many distinct values entered for it in the same column of the table). Multivalued attribute is depicted by a dual oval.
	Derived attribute	An attribute whose value is calculated (derived) from other attributes. The derived attribute may or may not be physically stored in the database. In the Chen notation, this attribute is represented by dashed oval.
Relationships		
	Strong relationship	A relationship where entity is existence-independent of other entities, and PK of Child doesn't contain PK component of Parent Entity. A strong relationship is represented by a single rhombus

	Weak (identifying) relationship	A relationship where Child entity is existence-dependent on parent, and PK of Child Entity contains PK component of Parent Entity. This relationship is represented by a double rhombus.

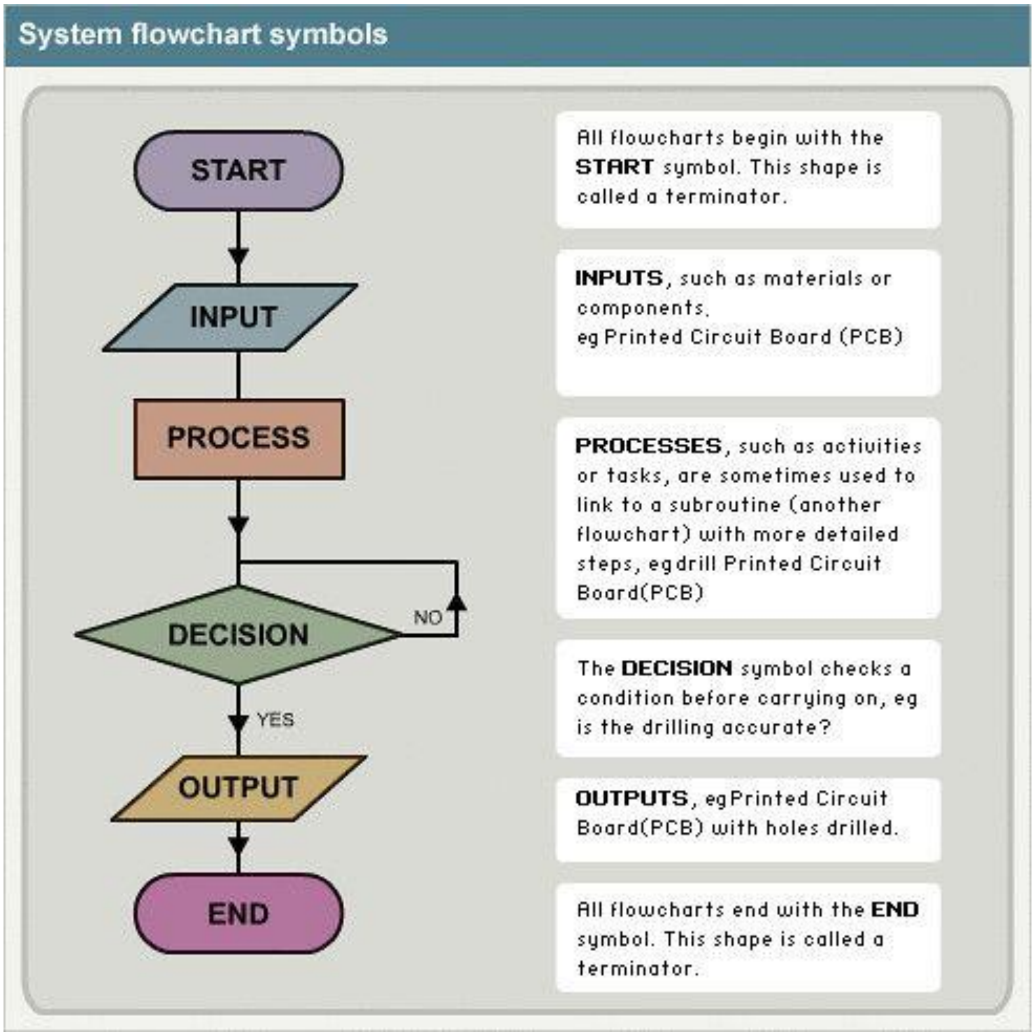
Entity Relationship Diagram Symbols — Crow's Foot notation

Symbol	Meaning
Relationships (Cardinality and Modality)	
	Zero or One
	One or More
	One and only One
	Zero or More
Many - to - One	
	a one through many notation on one side of a relationship and a one and only one on the other
	a zero through many notation on one side of a relationship and a one and only one on the other
	a one through many notation on one side of a relationship and a zero or one notation on the other
	a zero through many notation on one side of a relationship and a zero or one notation on the other
Many - to - Many	
	a zero through many on both sides of a relationship
	a zero through many on one side and a one through many on the other

	a one through many on both sides of a relationship
	a one and only one notation on one side of a relationship and a zero or one on the other
	a one and only one notation on both sides

Flow Charts

Flowcharts can be used to describe program logic, and are very useful in visualizing a modular design. A flowchart represents logical rules and interaction graphically, using a series of symbols connected by arrows. Using flowcharts, programmers can break large systems into subsystems and modules that are easier to understand and code



All flowcharts begin with the **START** symbol. This shape is called a terminator.

INPUTS, such as materials or components, eg Printed Circuit Board (PCB)

PROCESSES, such as activities or tasks, are sometimes used to link to a subroutine (another flowchart) with more detailed steps, eg drill Printed Circuit Board(PCB)

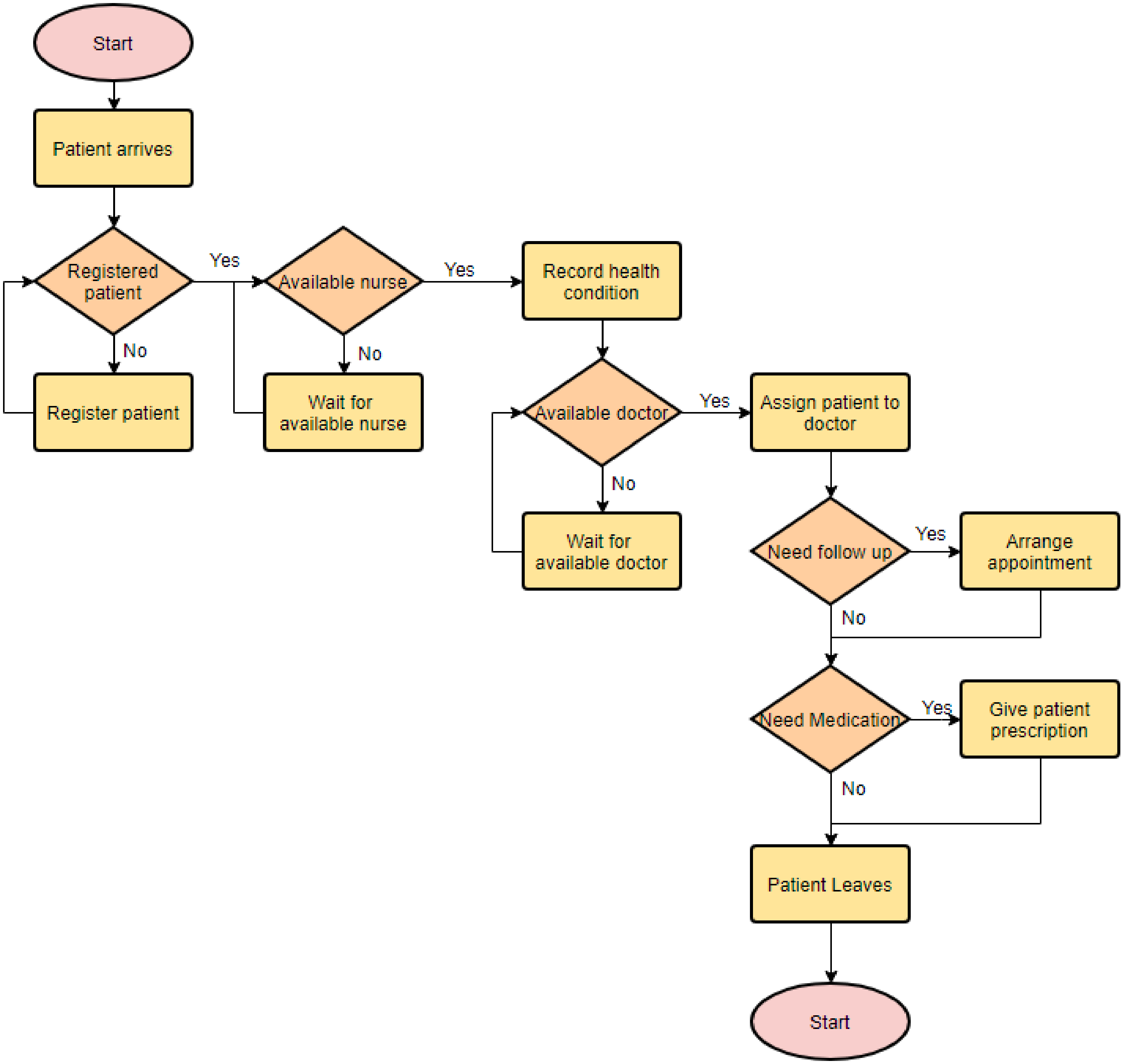
The **DECISION** symbol checks a condition before carrying on, eg is the drilling accurate?

OUTPUTS, eg Printed Circuit Board(PCB) with holes drilled.

All flowcharts end with the **END** symbol. This shape is called a terminator.

Flowchart Example – Medical Service

This is a hospital flowchart example that shows how clinical cases shall be processed. This flowchart uses decision shapes intensively in representing alternative flows.



Pseudo Code

Pseudo code is a technique for representing program logic. Pseudo code is similar to structured English. Pseudo code is not language-specific, so you can use it to describe a software module in plain English without requiring strict syntax rules. Using pseudo code, a systems analyst or a programmer can describe program actions that can be implemented in any programming language.

Decision Tables and Decision Trees

Decision tables and decision trees can be used to model business logic for an information system. In addition to being used as modeling tools, analysts and programmers can use decision tables and decision trees during system development, as they develop code modules that implement the logical rules.

Structured Application Development

Structured application development usually involves a top-down approach, which proceeds from a general design to a detailed structure. After a systems analyst documents the system's requirements, he or she breaks the system down into subsystems and modules in a process called partitioning. This approach also is called modular design and is similar to constructing a leveled set of DFDs. By assigning modules to different programmers, several development areas can proceed at the same time. Because all the modules must work together properly, an analyst must proceed carefully, with constant input from programmers and IT management to achieve a sound, well-integrated structure. The analyst also must ensure that integration capability is built into each design and thoroughly tested.

Structure Charts

Structure charts show the program modules and the relationships among them. A structure chart consists of rectangles that represent the program modules, with arrows and other symbols that provide additional information. Typically, a higher-level module, called a control module, directs lower-level modules, called subordinate modules. In a structure chart, symbols represent various actions or conditions. Structure chart symbols represent modules, data couples, control couples, conditions, and loops.

Module

A rectangle represents a module. Vertical lines at the edges of a rectangle indicate that module is a library module. A library module is reusable code and can be invoked from more than one point in the chart.

Data Couple

An arrow with an empty circle represents a data couple. A data couple shows data that one module passes to another.

Control Couple

An arrow with a filled circle represents a control couple. A control couple shows a message, also called a status flag, which one module sends to another. A module uses a flag to signal a specific condition or action to another module.

Condition

A line with a diamond on one end represents a condition. A condition line indicates that a control module determines which subordinate modules will be invoked, depending on a specific condition.

Loop

A curved arrow represents a loop. A loop indicates that one or more modules are repeated.

Cohesion and Coupling

Cohesion and coupling are important tools for evaluating the overall design. **Cohesion** measures a module's scope and processing characteristics. A module that performs a single function or task has a high degree of cohesion, which is desirable. Because it focuses on a single task, a cohesive module is much easier to code and reuse. If a module must perform multiple tasks, more complex coding is required and the module will be more difficult to create and maintain. If you need to make a module more cohesive, you can split it into separate units, each with a single function.

Coupling describes the degree of interdependence among modules. Modules that are independent are loosely coupled, which is desirable. Loosely coupled modules are easier to maintain and modify, because the logic in one module does not affect other modules. If a programmer needs to update a loosely coupled module, he or she can accomplish the task in a single location. If modules are tightly coupled, one module is linked to internal logic contained in another module.

Drawing a Structure Chart

If you used a structured analysis method, your structure charts will be based on the DFDs you created during data and process modeling. Typically, you follow four steps when you create a structure chart. You review DFDs to identify the processes and methods, identify the program modules and determine control-subordinate relationships, add symbols for couples and loops, and analyze the structure chart to ensure that it is consistent with your system documentation.

Step 1: review the DFDs

Your first step is to review all DFDs for accuracy and completeness, especially if changes have occurred since the systems analysis phase. If object models also were developed, you should analyze them to identify the objects, the methods that each object must perform, and the relationships among the objects. A method is similar to a functional primitive, and requires code to implement the necessary actions.

Step 2: Identify Modules and Relationships

Working from the logical model, you transform functional primitives or object methods into program modules. When analyzing a set of DFDs, remember that each DFD level represents a processing level. If you are using DFDs, you would work your way down from the context diagram to the lower level diagrams, identifying control modules and subordinate modules, until you reach functional primitives. If more cohesion is desired, you can divide processes into smaller modules that handle a single task.

Step 3: Add Couples, Loops, and Conditions

Next, you add couples, loops, and conditions to the structure chart. If you are working with DFDs, you can review the data flows and the data dictionary to identify the data elements that pass from one module to another. In addition to adding the data couples, you add control couples where a module is sending a control parameter, or flag, to another module. You also add loops and condition lines that indicate repetitive or alternative processing steps. If you also developed an object model, you can review the class diagrams and object relationship diagrams to be sure that you understand the interaction among the objects.

Step 4: Analyze the Structure Chart and the Data Dictionary

At this point, the structure chart is ready for careful analysis. You should check each process, data element, or object method to ensure that the chart reflects all previous documentation and that the logic is correct. You also should determine that modules are strongly cohesive and loosely coupled. Often, you must draw several versions of the chart. Some CASE tools can help you analyze the chart and identify problem areas.

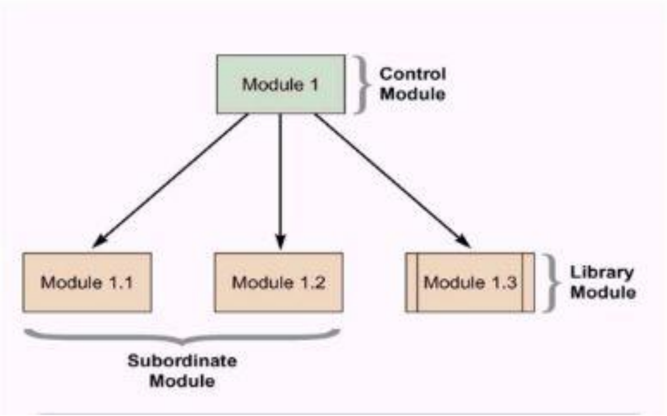
Structure Chart Symbols

Structure Chart symbols represent

- Modules (sequential logic)
- Conditions (decision logic)
- Loops (iterative logic)
- Data couples
- Control couples

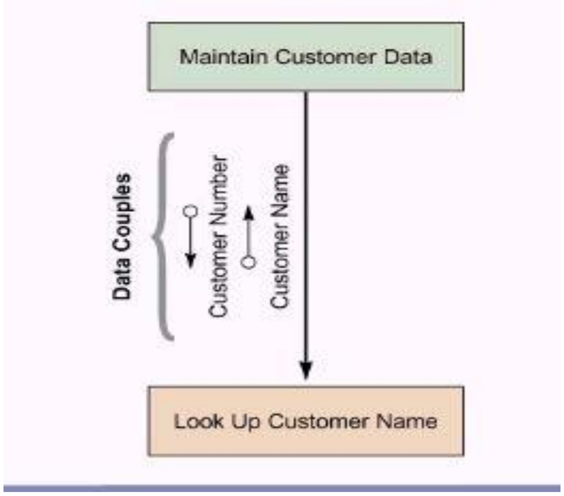
Structure Chart – Module

- Rectangle represents a module (program or subroutine)
- Control Modules (mainline) branch to sub-modules
- Library modules are reusable and can be invoked from more than one Control Module elsewhere in the system



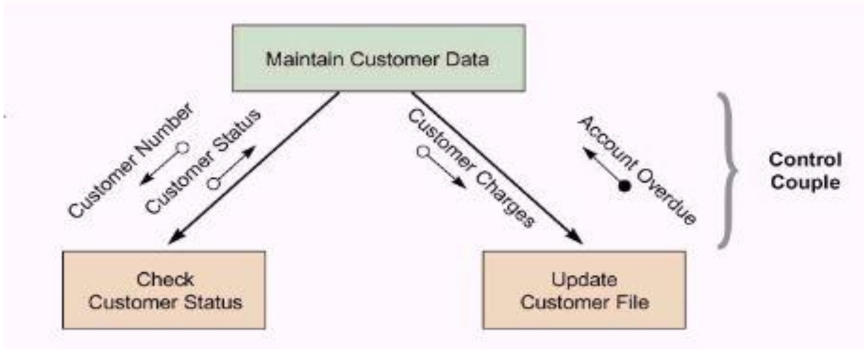
Structure Chart - Data Couple

- Arrow with an empty circle
- Shows the data one module passes to another



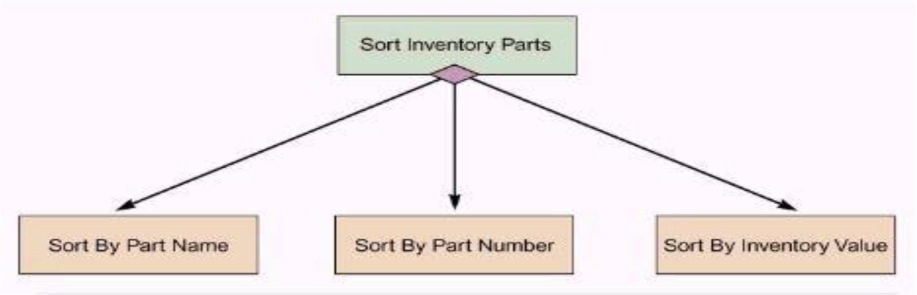
Structure Chart - Control Couple

- Arrow with filled circle
- Shows a message (flag) which one module sends to another
- Module uses a flag to signal a specific condition or action to another module



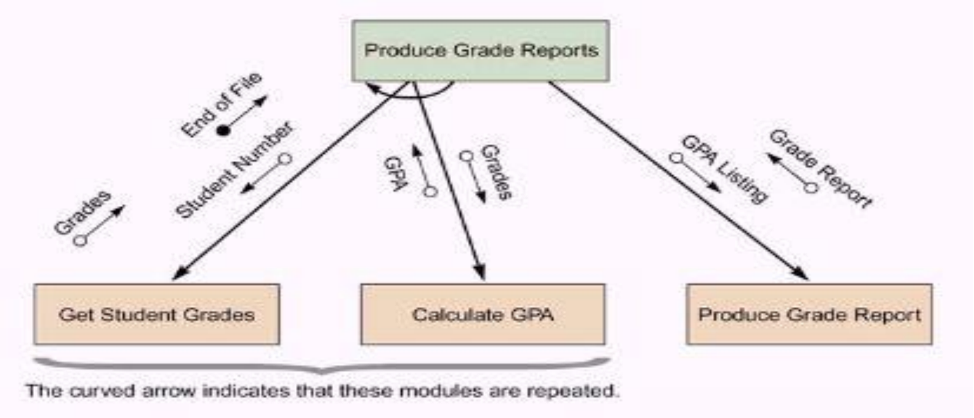
Structure Chart - Condition

- A line with a diamond
- Indicates that a control module determines which subordinate module will be invoked

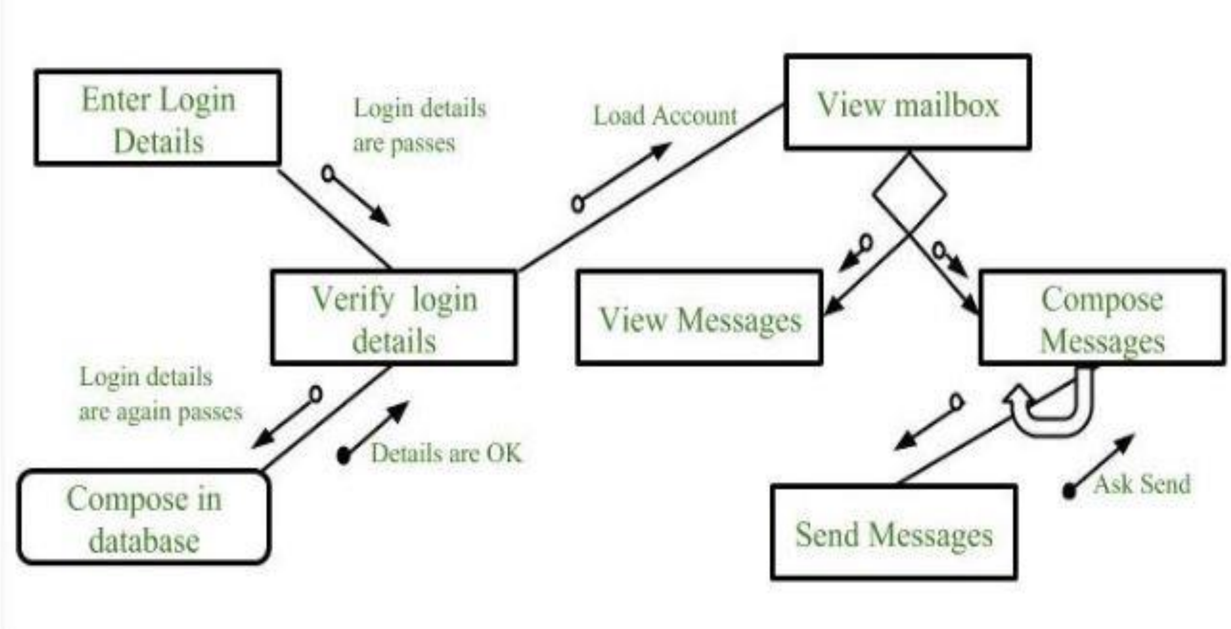


Structure Chart - Loop

- A curved arrow representing a loop
- One or more modules are repeated



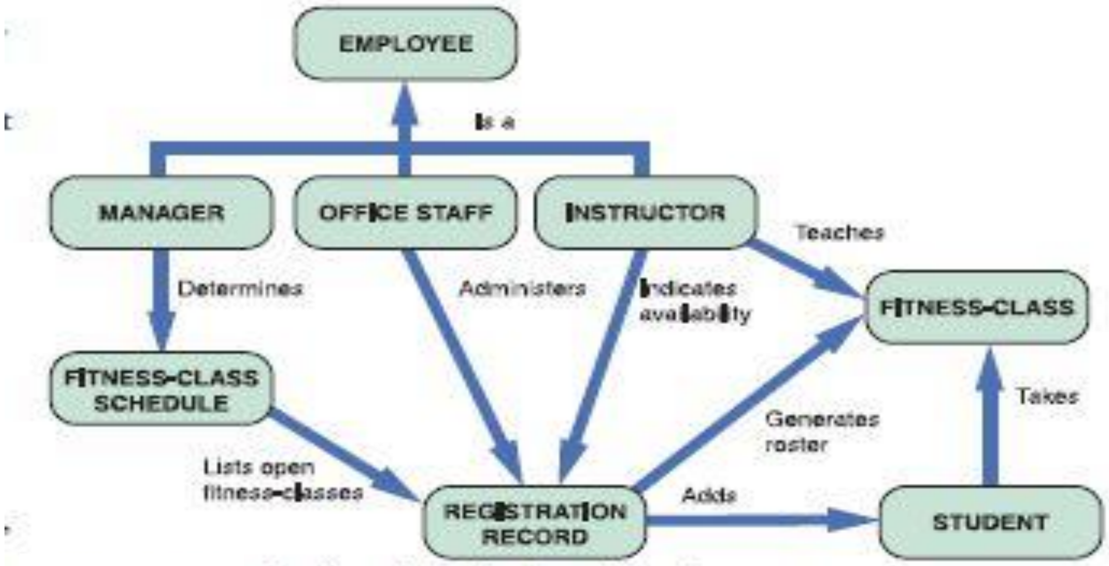
Example of a structure chart for a mail service



Object-Oriented Application Development

When implementing a structured design, a structure chart is used to describe the interaction between program modules, as explained earlier. In contrast, when implementing an object-oriented design, relationships between objects already exist. Because object interaction is defined during the O-O analysis process, the application's structure is represented by the object model itself. Objects contain both data and program logic, called methods. Individual object instances belong to classes of objects with similar characteristics. The relationship and interaction among classes are described using a class diagram. A class diagram includes the class **attributes**, which describe the characteristics of objects in the class, and **methods**, which represent program logic. In addition to class diagrams, programmers get an overview of object interaction by using object relationship diagrams that were developed during the O-O analysis process.

Properly implemented, object-oriented development can speed up projects, reduce costs, and improve overall quality. However, these results are not always achieved. Organizations sometimes have unrealistic expectations, and do not spend enough time learning about, preparing for, and implementing the OOD process. For example, no one would build a bridge without an analysis of needs, supporting data, and a detailed blueprint — and the bridge would not be opened for traffic until it had been carefully inspected and checked to ensure that all specifications were met. O-O software developers sometimes forget that the basic rules of architecture also apply to their projects. In summary, to secure the potential benefits of object-oriented development, systems analysts must carefully analyze, design, implement, test, and document their O-O projects.



An object-relationship diagram for a fitness center.

Implementation of Object-Oriented Designs

When a programmer translates an object-oriented design into an application, he or she analyzes the classes, attributes, methods, and messages that are documented in the object model. During this process, the programmer makes necessary revisions and updates to class diagrams, sequence diagrams, state transition diagrams, and activity diagrams. The programmer's main objective is to translate object methods into program code modules and determine what event or message will trigger the execution of each module. To accomplish the task, the programmer analyzes sequence diagrams and state transition diagrams that show the events and messages that trigger changes to an object. O-O applications are called event-driven, because each event, transaction, or message triggers a corresponding action. The programmer can represent the program steps in pseudo code initially, or use CASE tools and code generators to create object-oriented code directly from the object model.

Object-Oriented Cohesion and Coupling

The principles of cohesion and coupling also apply to object-oriented application development. Classes should be as loosely coupled (independent of other classes) as possible. In addition, an object's methods also should be loosely coupled (independent of other methods) and highly cohesive (perform closely related actions). By following these principles, classes and objects are easier to understand and edit. O-O programmers who ignore cohesion and coupling concepts may end up creating a web of code that is difficult to maintain. When code is scattered in various places, editing becomes complicated and expensive.

AGILE APPLICATION DEVELOPMENT

Agile development is a distinctly different systems development method. It shares many of the steps found in traditional development, but uses a highly iterative process. The development team is in constant communication with the **customer** or primary user, shaping and forming the system to match the customer's specifications. Agile development is aptly named because it is based on a quick and nimble development process that easily adapts to change. Agile development focuses on small teams, intense communication, and rapid development iterations. Extreme Programming (XP), which is one of the newest agile methods emphasizes customer satisfaction, teamwork, speed, and simplicity.

Extreme Programming uses an interesting concept called parallel programming. In **parallel programming**, two programmers work on the same task on the same computer; one drives (programs) while the other navigates (watches). The onlooker examines the code strategically to see the *forest* while the driver is concerned with the individual *trees* immediately in front of him or her. The two discuss their ideas continuously throughout the process. Another important concept in XP is that unit tests are designed *before* code is written. This **test-driven design** focuses on end results from the beginning and prevents programmers from straying from their goals. Because of the magnitude and intensity of the multi-cycle process, agile testing relies heavily on automated testing methods and software. Programmers can use popular agile-friendly languages such as Python, Ruby, and Perl. However, agile methods do not require a specific programming language, and programmers also use various object-oriented languages such as Java, C++, and C#.

Agile methodology is becoming very popular for software projects. Its supporters boast that it speeds up software development and delivers precisely what the customer wants, when the customer wants it, while fostering teamwork and empowering employees. However, there are drawbacks to this adaptive rather than predictive method. Critics of agile development often claim that because it focuses on quick iterations and fast releases, it lacks discipline and produces systems of questionable quality. In addition, agile methodology may not work as well for larger projects because of their complexity and the lack of focus on a well-defined end product. Before implementing agile development, the proposed system and development methods should be examined carefully. As experienced IT professionals know, a one-size- fits-all solution does not exist.