A Technical View of System Analysis and Design

TG5.1 Developing an IT Architecture
TG5.2 Overview of the Traditional Systems Development Life Cycle
TG5.3 Alternative Methods and Tools for Systems Development
TG5.4 Component-Based Development and Web Services
A SIX-STEP PROCESS

An IT architecture is a conceptual framework for the organization of the IT infrastructure and applications. It is a plan for the structure and integration of IT resources and applications in the organization.

Once the corporate strategy team or steering committee decides on potential applications, an architecture must be developed.

**Step 1. Business goals and vision.** This step, in which the system analyst reviews the relevant business goals and vision, is sometimes referred to as “business architecture.”

**Step 2. Information architecture.** In this step a company analyst defines the information necessary to fulfill the objectives of Step 1. Here, one should examine each objective and goal, identify the information currently available, and determine what new information is needed. All potential users need to be involved.

**Step 3. Data architecture.** Once you know what information must be processed, you need to determine a data architecture—that is, exactly what data you have and what you want to get from customers, including Web-generated data. Of special interest is the investigation of all data that flow within the organization and to and from your business partners.

The result of your investigation will probably show that data are everywhere, from data warehouses to mainframe files to Excel files on users’ PCs. You need to conduct an analysis of the data, understanding their use, and examine the need for new data. This is when you need to think about how to process these data and what tools to use.

**Step 4. Application architecture.** At this point, you define the components or modules of the applications that will interface with the required data defined in Step 3. In this step you will build the conceptual framework of an application, but not the infrastructure that will support it. An example is shown in Figure TG5.1.

Many vendors, such as IBM, Oracle, and Microsoft, offer sophisticated IT application platforms that can significantly reduce the amount of code that programmers need to write. These application platforms also explain how the application should be structured.

Other factors that must be considered are scalability, security, the number and size of servers, and the networks. The need to interface with legacy systems and with sales, ERP, accounting, and human resources data must be considered. In addition, the ability to read real-time data is also important.

The major output of this step is to define the software components that meet the data requirements.

![Figure TG5.1 Architecture of an online travel agency.](image-url)
Step 5. Technical architecture. During the previous steps, designers informally considered the technical requirements. In this step, they must formally examine the specific hardware and software required to support the analysis in the previous steps. An inventory of the existing information resources is made, and an evaluation of the necessary upgrades and acquisitions is performed.

At this stage, designers must also examine the middleware needed for the application. EC applications require a considerable amount of transaction processing software. The more scalability and availability required, the more you need to invest in additional application servers and other hardware and software.

When selecting a programming language, designers may consider Java, Visual Studio, C11, CGI, and even COBOL, depending on the application. Also in this step, the operating systems, transaction processors, and networking devices required to support the applications must be decided on. Obviously, you want to leverage your existing IT resources, but this may not be the optimal approach.

Step 6. Organizational architecture. An organizational architecture deals with the human resources and procedures required by Steps 1 through 5. At this point, the legal, administrative, and financial constraints should be examined. For example, a lack of certain IT skills on your team may require hiring or retraining. Partial outsourcing may be a useful way to deal with skill deficiencies.

In the worst-case scenario, you outsource the entire job, but you can give the architecture to the vendor as a starting point. Also, vendor selection can be improved if the architectures (business, information, data, application, and technical) are considered.

Creating IT architecture may be a lengthy process, but it is necessary to go through it. You may want to develop metrics to help you to track the effectiveness of your IT architecture, and you certainly need to document the process and output of each step.

Once the IT architecture has been decided on, a development strategy can be formulated.

CONCLUSION

The systems development life cycle (SDLC) is the traditional systems development method used by organizations for large IT projects such as IT infrastructure. The SDLC is a structured framework that consists of sequential processes by which information systems are developed. As shown in Figure TG5.2, these processes include investigation, analysis, design, programming, testing, implementation, operation, and maintenance. The processes, in turn, consist of well-defined tasks. Large projects typically require all of the tasks, whereas smaller development projects may require only a subset of the tasks.

Other models for the SDLC may contain more or fewer than the eight stages we present here. The flow of tasks, however, remains largely the same, regardless of the number of stages. In the past, developers used the waterfall approach to the SDLC, in which tasks in one stage were completed before the work proceeded to the next stage. Today, systems developers go back and forth among the stages as necessary.

Within the waterfall approach, there is an iterative feature. Iteration is the revising of the results of any development process when new information makes this revision desirable. Iteration does not mean that developments should be subjected to infinite revisions, which would never allow systems to be implemented and utilized. It does mean that developers must evaluate any new development information they come across to determine whether it warrants causing revisions to the existing development. It is especially important for e-commerce development because EC systems must be constantly evolving to meet new demands of their users and to stay ahead of the competition.
Systems development projects produce desired results through team efforts. Development teams typically include users, systems analysts, programmers, and technical specialists. Users are employees from all functional areas and levels of the organization who will interact with the system, either directly or indirectly. Systems analysts are information systems professionals who specialize in analyzing and designing information systems. Programmers are information systems professionals who modify existing computer programs or write new computer programs to satisfy user requirements. Technical specialists are experts on a certain type of technology, such as databases or telecommunications. All people who are affected by changes in information systems (e.g., users and managers) are known as systems stakeholders, and are typically involved by varying degrees and at various times in the systems development.

In the remainder of this section, we will look at each of the processes (phases) in the eight-stage SDLC.

**Systems Investigation**

Systems development professionals agree that the more time invested in understanding the business problem to be solved, in understanding technical options for systems, and in understanding problems that are likely to occur during development, the greater the chance of successfully solving the problem. For these reasons, systems investigation begins with the business problem (or business opportunity).

Problems (and opportunities) often require not only understanding them from the internal point of view, but also seeing them as organizational partners (suppliers or customers) would see them. Another useful perspective is that of competitors. (How have they responded to similar situations, and what outcomes and additional opportunities have materialized?) Creativity and out-of-the-box thinking can pay big dividends when isolated problems can be recognized as systemic failures whose causes cross organizational boundaries. Once these perspectives can be gained, those involved can also begin to better see the true scope of the project and propose possible solutions. Then, an initial assessment of these proposed system solutions can begin.

**Feasibility Studies.** The next task in the systems investigation stage is the feasibility study. The feasibility study determines the probability of success of the proposed project and provides a rough assessment of the project’s technical, economic, organizational, and behavioral feasibility. The feasibility study is critically important to
the systems development process because done properly the study can prevent organizations from making costly mistakes (such as creating systems that will not work, that will not work efficiently, or that people cannot or will not use). The various feasibility analyses also give the stakeholders an opportunity to decide what metrics to use to measure how a proposed system (and later, a completed system) meets their various objectives.

**Technical Feasibility.** Technical feasibility determines if the hardware, software, and communications components can be developed and/or acquired to solve the business problem. Technical feasibility also determines if the organization’s existing technology can be used to achieve the project’s performance objectives.

**Economic Feasibility.** Economic feasibility determines if the project is an acceptable financial risk and if the organization can afford the expense and time needed to complete the project. Economic feasibility addresses two primary questions: Do the benefits outweigh the costs of the project? Can the project be completed as scheduled?

Three commonly used methods to determine economic feasibility are return on investment (ROI), net present value (NPV), and breakeven analysis. **Return on investment** is the ratio of the net income attributable to a project divided by the average assets invested in the project. The **net present value** is the net amount by which project benefits exceed project costs, after allowing for the cost of capital and the time value of money. **Breakeven analysis** determines the point at which the cumulative cash flow from a project equals the investment made in the project.

Determining economic feasibility in IT projects is rarely straightforward, but it often is essential. Part of the difficulty stems from the fact that benefits often are intangible. Another potential difficulty is that the proposed system or technology may be “cutting edge,” and there may be no previous evidence of what sort of financial payback is to be expected.

**Organizational Feasibility.** Organizational feasibility has to do with an organization’s ability to accept the proposed project. Sometimes, for example, organizations cannot accept a financially acceptable project due to legal or other constraints. In checking organizational feasibility, one should consider the organization’s policies and politics, including impacts on power distribution, business relationships, and internal resources availability.

**Behavioral Feasibility.** Behavioral feasibility addresses the human issues of the project. All systems development projects introduce change into the organization, and people generally fear change. Overt resistance from employees may take the form of sabotaging the new system (e.g., entering data incorrectly) or deriding the new system to anyone who will listen. Covert resistance typically occurs when employees simply do their jobs using their old methods.

A more positive and pragmatic concern of behavioral feasibility is assessing the skills and training needs that often accompany a new information system. In some organizations, a proposed system may require mathematical or linguistic skills beyond what the workforce currently possesses. In others, a workforce may simply need additional skill building rather than remedial education. Behavioral feasibility is as much about “can they use it” as it is about “will they use it.”

After the feasibility analysis, a “Go/No-Go” decision is reached. The functional area manager for whom the system is to be developed and the project manager sign off on the decision. If the decision is “No Go,” the project is put on the shelf until conditions are more favorable, or the project is discarded. If the decision is “Go,” then the systems development project proceeds and the systems analysis phase begins.

**SYSTEMS ANALYSIS**

The systems analysis stage produces the following information: (1) strengths and weaknesses of the existing system; (2) functions that the new system must have to solve the business problem; and (3) user information requirements for the new system. Armed with this information, systems developers can proceed to the systems design stage.
There are two main approaches in systems analysis: the traditional (structured) approach, and the object-oriented approach. The traditional approach emphasizes “how,” whereas the object-oriented approach emphasizes “what.”

Systems analysis describes what a system must do to solve the business problem, and systems design describes how the system will accomplish this task. The deliverable of the systems design phase is the technical design that specifies the following:

- System outputs, inputs, and user interfaces
- Hardware, software, databases, telecommunications, personnel, and procedures
- How these components are integrated

This output represents the set of system specifications.

Systems design encompasses two major aspects of the new system: **Logical system design** states what the system will do, using abstract specifications. **Physical system design** states how the system will perform its functions, with actual physical specifications. Logical design specifications include the design of outputs, inputs, processing, databases, telecommunications, controls, security, and IS jobs. Physical design specifications include the design of hardware, software, database, telecommunications, and procedures. For example, the logical telecommunications design may call for a wide-area network connecting the company’s plants. The physical telecommunications design will specify the types of communications hardware (e.g., computers and routers), software (e.g., the network operating system), media (e.g., fiber optics and satellite), and bandwidth (e.g., 100 Mbps).

When both of these aspects of system specifications are approved by all participants, they are “frozen.” That is, once the specifications are agreed upon, they should not be changed. However, users typically ask for added functionality in the system (called scope creep). This occurs for several reasons: First, as users more clearly understand how the system will work and what their information and processing needs are, they see additional functions that they would like the system to have. Also, as time passes after the design specifications are frozen, business conditions often change, and users ask for added functionality. Because scope creep is expensive, project managers place controls on changes requested by users. These controls help to prevent runaway projects—systems development projects that are so far over budget and past deadline that they must be abandoned, typically with large monetary loss.

Programming involves the translation of the design specifications into computer code. This process can be lengthy and time-consuming, because writing computer code remains as much an art as a science. Large systems development projects can require hundreds of thousands of lines of computer code and hundreds of computer programmers. In such projects, programming teams are used. These teams often include functional-area users to help the programmers focus on the business problem at hand.

In an attempt to add rigor (and some uniformity) to the programming process, programmers use structured programming techniques. These techniques improve the
logical flow of the program by decomposing the computer code into modules, which
are sections of code (subsets of the entire program). This modular structure allows
for more efficient and effective testing because each module can be tested by itself.
The structured programming techniques include the following restrictions:

- Each module has one, and only one, function.
- Each module has only one entrance and one exit. That is, the logic in the computer
  program enters a module in only one place and exits in only one place.
- GO TO statements are not allowed.

For example, a flowchart for a simple payroll application might look like the one
shown in Figure TG5.3. The figure shows the only three types of structures that are
used in structured programming: sequence, decision, and loop. In the sequence struc-
ture, program statements are executed one after another until all of the statements
in the sequence have been executed. The decision structure allows the logic flow to
branch, depending on certain conditions being met. The loop structure enables the
software to execute the same program, or parts of a program, until certain conditions
are met (e.g., until the end of the file is reached, or until all records have been
processed).

As already noted, structured programming enforces some standards about how
program code is written. This approach and some others were developed not only
to improve programming, but also to standardize how a firm’s various programmers
do their work. This uniform approach helps ensure that all of the code developed by
different programmers will work together. Even with these advances, however, pro-
gramming can be difficult to manage.
Thorough and continuous testing occurs throughout the programming stage. Testing checks to see if the computer code will produce the expected and desired results under certain conditions. Testing requires a large amount of time, effort, and expense to do properly. However, the costs of improper testing, which could possibly lead to a system that does not meet its objectives, are enormous.

Testing is designed to detect errors (“bugs”) in the computer code. These errors are of two types: syntax errors and logic errors. Syntax errors (e.g., a misspelled word or a misplaced comma) are easier to find and will not permit the program to run. Logic errors permit the program to run but result in incorrect output. Logic errors are more difficult to detect because the cause is not obvious. The programmer must follow the flow of logic in the program to determine the source of the error in the output.

To have a systematic testing of the system, we must start with a comprehensive test plan. There are several types of testing: unit testing, in which each module is tested alone in an attempt to discover any errors in its code. String testing puts together several modules, to check the logical connection among them. The next level, integration testing, brings together various programs for testing purposes. System testing brings together all of the programs that comprise the system.

As software increases in complexity, the number of errors increases, making it almost impossible to find them all. This situation has led to the idea of “good-enough” software, software that developers release knowing that errors remain in the code but believing that the software will still meet its functional objectives. That is, they have found all the “show-stopper” bugs, errors that will cause the system to shut down or will cause catastrophic loss of data.

Implementation (or deployment) is the process of converting from the old system to the new system. Organizations use four major conversion strategies: parallel, direct, pilot, and phased.

In a parallel conversion, the old system and the new system operate simultaneously for a period of time. That is, both systems process the same data at the same time, and the outputs are compared. This type of conversion is the most expensive, but also the least risky. Most large systems have a parallel conversion process to lessen the risk.

In a direct conversion, the old system is cut off and the new system is turned on at a certain point in time. This type of conversion is the least expensive, but the most risky if the new system doesn’t work as planned. Few systems are implemented using this type of conversion, due to the risk involved.

A pilot conversion introduces the new system in one part of the organization, such as in one plant or in one functional area. The new system runs for a period of time and is assessed. After the new system works properly, it is introduced in other parts of the organization.

A phased conversion introduces components of the new system, such as individual modules, in stages. Each module is assessed, and, when it works properly, other modules are introduced until the entire new system is operational.

Enterprise application integration (EAI) is often called the middleware. Interfaces were developed to map the major packages to a single conceptual framework that guides what all these packages do and the kinds of information they normally need to share. This conceptual framework could be used to translate the data and processes from each vendor’s package to a common language. It is the only way to implement collaborative supply chain sharing of information.

XML is the technology that is being used by many EAI vendors in their cross-enterprise application development. It can be thought of as a way for providing variable format messages that can be shared between any two computer systems, as long as they both understand the format (tags) that is (are) being used.

After conversion, the new system will operate for a period of time, until (like the old system it replaced) it no longer meets its objectives. Once the new system’s opera-
tions are stabilized, audits are performed during operation to assess the system’s capabilities and determine if it is being used correctly.

Systems need several types of maintenance. The first type is debugging the program, a process that continues throughout the life of the system. The second type is updating the system to accommodate changes in business conditions. An example would be adjusting to new governmental regulations (such as tax rate changes). These corrections and upgrades usually do not add any new functionality; they are necessary simply in order for the system to continue meeting its objectives. The third type of maintenance adds new functionality to the system—adding new features to the existing system without disturbing its operation.

The **prototyping** approach defines an initial list of user requirements, builds a prototype system, and then improves the system in several iterations based on users’ feedback. Developers do not try to obtain a complete set of user specifications for the system at the outset and do not plan to develop the system all at once. Instead, they quickly develop a prototype, which either contains parts of the new system of most interest to the users or is a small-scale working model of the entire system. Users make suggestions for improving the prototype, based on their experiences with it.

The developers then review the prototype with the users and use the suggestions to refine the prototype. This process continues through several iterations until either the users approve the system or it becomes apparent that the system cannot meet users’ needs. If the system is viable, the developers can use the prototype on which to build the full system. Developing screens that a user will see and interact with is a typical use of prototyping. (See Figure TG5.4 for a model that shows the prototyping process.)

The main advantage of prototyping is that it speeds up the development process. In addition, prototyping gives users the opportunity to clarify their information requirements as they review iterations of the new system. Prototyping is especially useful in the development of decision support systems and executive information systems, where user interaction is particularly important.

Prototyping also has disadvantages. Because it can largely replace the analysis and design stages of the SDLC in some projects, systems analysts may not produce adequate documentation for the programmers. This lack of documentation can lead to problems after the system becomes operational and needs maintenance. In addition, prototyping can result in an excess of iterations, which can consume the time that prototyping should be saving.
Inside spiral development there is prototyping. The prototype is a model of a system that can be used to communicate the requirements and design of that part of the system between developers and their clients.

**Joint application design (JAD)** is a group-based tool for collecting user requirements and creating system designs. JAD is most often used within the systems analysis and systems design stages of the SDLC.

In the traditional SDLC, systems analysts interview or directly observe potential users of the new information system individually to understand each user’s needs. The analysts will obtain many similar requests from users, but also many conflicting requests. The analysts must then consolidate all requests and go back to the users to resolve the conflicts, a process that usually requires a great deal of time. In contrast, JAD has a group meeting in which all users meet simultaneously with analysts. It is basically a group decision-making process and can be computerized or done manually. During this meeting, all users jointly define and agree upon systems requirements. This process saves a tremendous amount of time. e-JAD is an extension of JAD whereby the group meeting is done remotely using groupware software.

The JAD approach to systems development has several advantages. First, the group process involves many users in the development process while still saving time. This involvement leads to greater support for the new system and can produce a system of higher quality. This involvement also may lead to easier implementation of the new system and lower training costs.

The JAD approach also has disadvantages. First, it is very difficult to get all users to the JAD meeting. For example, large organizations may have users literally all over the world. Second, the JAD approach has all of the problems caused by any group process (e.g., one person can dominate the meeting, some participants may not contribute in a group setting). To alleviate these problems, JAD sessions usually have a facilitator, who is skilled in systems analysis and design as well as in managing group meetings and processes. Also, the use of groupware (such as GDSS) can help facilitate the meeting.
Rapid application development (RAD) is a systems development method that can combine JAD, prototyping, and integrated CASE tools (described next) to rapidly produce a high-quality system. Initially, JAD sessions are used to collect system requirements so that users are intensively involved early on. The development process in RAD is iterative, similar to prototyping, in which requirements, designs, and the system itself are developed with sequential refinements. However, RAD and prototyping use different tools. Prototyping typically uses specialized languages, such as fourth-generation languages (4GLs), Web-based development tools, and screen generators; RAD uses ICASE tools (discussed next) to quickly structure requirements and develop prototypes. As the prototypes are developed and refined, users review them in additional JAD sessions. RAD produces functional components of a final system, rather than limited-scale versions. For more details, see Figure TG5.5. The figure also compares RAD to SDLC.

Rapid application development (RAD) methodologies and tools make it possible to develop systems faster, especially systems where the user interface is an important component. RAD can also improve the process of rewriting legacy applications.

Computer-aided software engineering (CASE) is a development approach that uses specialized tools to automate many of the tasks in the SDLC. The tools used to automate the early stages of the SDLC (systems investigation, analysis, and design) are called upper CASE tools. The tools used to automate later stages in the SDLC (programming, testing, operation, and maintenance) are called lower CASE tools. CASE tools that provide links between upper CASE and lower CASE tools are called integrated CASE (ICASE) tools. Some CASE tools can even work backward, modifying the model after modifying the coding. See, for example, IBM’s Rational Rose.

CASE tools provide advantages for systems developers. These tools can produce systems with a longer effective operational life that more closely meet user requirements. CASE tools can speed up the development process and result in systems that are more flexible and adaptable to changing business conditions. Finally, systems produced using CASE tools typically have excellent documentation.

On the other hand, CASE tools can produce initial systems that are more expensive to build and maintain. CASE tools do require more extensive and accurate definition of user needs and requirements. Also, CASE tools are difficult to customize and may be difficult to use with existing systems.

Object-oriented development is based on a fundamentally different view of computer systems than that found in traditional SDLC development approaches. Traditional approaches provide specific step-by-step instructions in the form of computer programs, in which programmers must specify every procedural detail. These programs usually result in a system that performs the original task but may not be suited for handling other tasks, even when the other tasks involve the same real-world entities.
For example, a billing system will handle billing but probably will not be adaptable to handle mailings for the marketing department or generate leads for the sales force, even though the billing, marketing, and sales functions all use similar data (e.g., customer names, addresses, and purchases). An object-oriented (OO) system, on the other hand, begins not with the task to be performed, but with the aspects of the real world that must be modeled to perform that task. Therefore, in the example above, if the firm has a good model of its customers and its interactions with them, this model can be used equally well for billings, mailings, and sales leads.

The object-oriented (OO) approach to software development offers many advantages:

- It reduces the complexity of systems development and leads to systems that are easier and quicker to build and maintain, because each object is relatively small and self-contained.
- It improves programmers’ productivity and quality. Once an object has been defined, implemented, and tested, it can be reused in other systems.
- Systems developed with the OO approach are more flexible. These systems can be modified and enhanced easily by changing some types of objects or by adding new types.
- The OO approach allows the systems analyst to think at the level of the real-world systems (as users do), rather than at the level of the programming language. The basic operations of an enterprise change much more slowly than the information needs of specific groups or individuals. Therefore, software based on generic models (which the OO approach is) will have a longer life span than programs written to solve specific, immediate problems.
- The OO approach is also ideal for developing Web applications.
- The OO approach depicts the various elements of an information system in user terms (i.e., business or real-world terms), and therefore, the users have a better understanding of what the new system does and how it meets its objectives.

The OO approach does have some disadvantages: OO systems, especially those written in Java, generally run more slowly than those developed in other programming languages. Also, many programmers have little skill and experience with OO languages, necessitating retraining.

An object-oriented development environment provides a framework that encourages designers to think in object-oriented terms, to design systems with conceptual integrity and clear separation of function from internal implementation. It also provides substantial assistance to the developer in automating the production of executable software from the object-oriented model. Interface logic, and the underlying middleware, are generated by the component-based development environment.

**Object-Oriented Analysis and Design.** The development process for an object-oriented system begins with a feasibility study and analysis of the existing system. Systems developers identify the objects in the new system—the fundamental elements in OO analysis and design. Each object represents a tangible real-world entity, such as a customer, bank account, student, or course. Objects have properties. For example, a customer has an identification number, name, address, account number(s), and so on. Objects also contain the operations that can be performed on their properties. For example, customer objects’ operations may include obtain-account-balance, open-account, withdraw-funds, and so on.

Therefore, object-oriented analysts define all of the relevant objects needed for the new system, including their properties (called data values) and their operations (called behaviors). They then model how the objects interact to meet the objectives of the new system. In some cases, analysts can reuse existing objects from other applications (or from a library of objects) in the new system, saving time spent coding. In most cases, however, even with object reuse, some coding will be necessary to customize the objects and their interactions for the new system.

Comparison of the various development methods is shown in Table TG5.1.
### TABLE TG5.1 Advantages and Disadvantages of Systems Acquisition Methodologies

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Traditional Systems Development (SDLC)</strong></td>
<td>• Forces staff to be systematic by going through every step in a structured process.</td>
<td>• May produce excessive documentation.</td>
</tr>
<tr>
<td></td>
<td>• Enforces quality by maintaining standards.</td>
<td>• Users are often unwilling or unable to study the specifications they approve.</td>
</tr>
<tr>
<td></td>
<td>• Has lower probability of missing important issues in collecting user requirements.</td>
<td>• Takes too long to go from the original ideas to a working system.</td>
</tr>
<tr>
<td><strong>Prototyping</strong></td>
<td>• Helps clarify user requirements.</td>
<td>• Users have trouble describing requirements for a proposed system.</td>
</tr>
<tr>
<td></td>
<td>• Helps verify the feasibility of the design.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Promotes genuine user participation in the development process.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Promotes close working relationship between systems developers and users.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Works well for ill-defined problems.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• May produce part of the final system.</td>
<td></td>
</tr>
<tr>
<td><strong>Joint Application Development (JAD)</strong></td>
<td>• Easy for senior management to understand.</td>
<td>• Difficult and expensive to get all people to the same place at the same time.</td>
</tr>
<tr>
<td></td>
<td>• Provides needed structure to the user requirements collection process.</td>
<td>• Potential to have dysfunctional groups.</td>
</tr>
<tr>
<td><strong>Rapid Application Development (RAD)</strong></td>
<td>• Active user involvement in analysis and design stages.</td>
<td>• System often narrowly focused, which limits future evolution, flexibility, and adaptability to changing business conditions.</td>
</tr>
<tr>
<td></td>
<td>• Easier implementation due to user involvement.</td>
<td>• System may be built quickly, which may result in lower quality.</td>
</tr>
<tr>
<td><strong>Object-Oriented Development (OO)</strong></td>
<td>• Integration of data and processing during analysis and design should lead to higher-quality systems.</td>
<td>• Very difficult to train analysts and programmers on the OO approach.</td>
</tr>
<tr>
<td></td>
<td>• Reuse of common objects and classes makes development and maintenance easier.</td>
<td>• Limited use of common objects and classes.</td>
</tr>
<tr>
<td><strong>End-User Development</strong></td>
<td>• Bypasses the information systems department and avoids delays.</td>
<td>• Creates lower-quality systems because an amateur does the programming.</td>
</tr>
<tr>
<td></td>
<td>• User controls the application and can change it as needed.</td>
<td>• May eventually require consulting and maintenance assistance from the IT department.</td>
</tr>
<tr>
<td></td>
<td>• Directly meets user requirements.</td>
<td>• System may not have adequate documentation.</td>
</tr>
<tr>
<td></td>
<td>• Increased user acceptance of new system.</td>
<td>• Poor quality control.</td>
</tr>
<tr>
<td></td>
<td>• Frees up IT resources and may reduce application development backlog.</td>
<td>• System may not have adequate interfaces to existing systems.</td>
</tr>
<tr>
<td><strong>External Acquisition (Buy or Lease)</strong></td>
<td>• Software exists and can be tried out.</td>
<td>• Controlled by another company that has its own priorities and business considerations.</td>
</tr>
<tr>
<td></td>
<td>• Software has been used for similar problems in other organizations.</td>
<td>• Package’s limitations may prevent desired business processes.</td>
</tr>
<tr>
<td></td>
<td>• Reduces time spent for analysis, design, and programming.</td>
<td>• May be difficult to get needed enhancements if other companies using the package do not need those enhancements.</td>
</tr>
<tr>
<td></td>
<td>• Has good documentation that will be maintained.</td>
<td>• Lack of intimate knowledge about how the software works and why it works that way.</td>
</tr>
</tbody>
</table>
An information systems development methodology (ISDM) can be defined as a collection of procedures, techniques, tools, and documentation aids that help systems developers in their efforts to implement a new information system. The methodology consists of phases, themselves consisting of subphases, which guide the systems developers in their choice of the techniques that might be appropriate at each stage of the project, and also help them plan, manage, control, and evaluate information systems projects.

A methodology is a set of practices and procedures, with supporting templates and knowledge bases, that systematically organizes the development process. (A methodology is different from a method.) A methodology should specify the training needs of the users and specifically address the critical issue of development philosophy. The objectives of using a methodology are (1) a better end product, (2) a better development process, and (3) a standardized process.

Different methodologies make different assumptions about the business and work environments of the project, and knowing each of their pros and cons allows a team to pick the most efficient methodology for its particular project. Some methodologies emphasize testing, some documentation; others stress code reusability. Certain methodologies are better suited for projects with tight deadlines or unclear and changing requirements.

Executing against a methodology reduces the knowledge and experience required by a development team. However, the team needs to learn the rules and practices of a specific methodology.

Techniques and Tools Features in Each Methodology. A technique is a way of doing a particular activity in the information systems development process, and any particular methodology may recommend techniques to carry out many of these activities. Techniques include holistic, data, process, object-oriented, project management, organizational, and people.

Each technique may involve the use of one or more tools that represent some of the artifacts used in information systems development. Tools include groupware (e.g., GroupSystems), Web site development (e.g., DreamWeaver), drawing (e.g., Microsoft Visio), project management (e.g., Microsoft Project), and database management (e.g., Microsoft Access). Tools used in development can range from simple automation (e.g., a drawing program such as Visio) to fully featured modeling tools such as Rational Rose, which is capable of interfacing to a repository through XML to share data with other tools in a cooperative total development environment.

Component-Based Development and Web Services

Object-oriented development, discussed in Section TG5.3, has its downside: business objects, though they represent things in the real world, can become unwieldy when they are combined and recombined in large-scale commercial applications. What is needed, instead, are suites of business objects that provide major chunks of application functionality (e.g., preprogrammed workflow, order placing) that can be “snapped together” to create complete business applications.

This approach is embodied in component-based development (CBD), the upcoming evolutionary step beyond object-oriented development. CBD uses preprogrammed components to develop applications. For the purposes of independent deployment, a component needs to be a binary unit.

A component’s functionality can be accessed only through its interfaces. Components must have software “plug points” that fit into sockets provided by a component execution environment. The component execution environment is required to provide run-time technical infrastructure services and to hide low-level technology issues from the business solution developer.
Rather than elicit synchronous interactions between components, a component invokes an operation in another component by sending a message. Where integration is needed across architectural domains, loosely coupled integration is more appropriate than a tightly coupled arrangement. In a tightly coupled integration, a component needs to know the name of the service it wants to call. In a loosely coupled integration with a message broker, an application makes its request by sending a message, in proper standard format, to the message broker. Based on the message content, the message broker forwards the message to the application that accepts the message and acts upon it.

**Key Characteristics of Components in Component-Based Development.** Components used in distributed computing need to possess several key characteristics to work correctly, and they can be viewed as an extension of the object-oriented paradigm. The two main traits borrowed from the world of object-oriented technology are encapsulation and data hiding.

Components encapsulate the routines or programs that perform discrete functions. In a component-based program, one can define components with various published interfaces. One of these interfaces might be, for example, a data-comparison function. If this function is passed to two data objects to compare, it returns the results. All manipulations of data are required to use the interfaces defined by the data object, so the complete function is encapsulated in this object, which has a distinct interface to other systems. Now, if the function has to be changed, only the program code that defines the object must be changed, and the behavior of the data comparison routine is updated immediately, a feature known as encapsulation.

Data hiding addresses a different problem. It places data needed by a component object’s functions within the component, where it can be accessed only by specially designated functions in the component itself. Data hiding is a critical trait of distributed components. The fact that only designated functions can access certain data items, and outside “requestors” have to query the component, simplifies maintenance of component-oriented programs.

**Component-Based Development of E-Commerce Applications.** Plug-and-play business application components can be “glued together” rapidly to develop complex distributed applications, such as those needed for e-commerce. Component-based EC development is gaining momentum. It is supported by Microsoft and the Object Management Group (OMG), which have put in place many of the standards needed to make component-based development a reality. There are several methods that developers can use for integrating components. A logical architecture for component-based development of e-commerce applications can be described in layers, as shown in Figure TG5.6.
The Role of Component-Based Approach in Software Reuse. The efficient development of software reuse has become a critical aspect in the overall IS strategies of many organizations. An increasing number of companies have reported reuse successes. The traditional reuse paradigm allows changes to the code that is to be reused (“white-box reuse”). Component-based software development advocates that components are reused as is (“black-box reuse”). Taking the black-box reuse concept one step further is the idea of leveraging existing software using Web Services (our next topic). Both component-based development and Web Services are receiving growing interest among members of the IS community.

The major application of Web Services is systems integration. Applications need to be integrated with databases and with other applications. Users need to interface with the data warehouse to conduct analysis, and almost any new system needs to be integrated with older ones. Finally, the increase of B2B and e-busines activities requires the integration of application and databases of business partners (external integration). Because Web Services can contribute so much to systems integration, their use is growing rapidly.

The original term for Web Services was “application services.” They are services that are made available from a business’s server for Web users. Because of their great interoperability and extensibility (due to the use of XML), Web Services can be combined in a loosely coupled way in order to achieve complex operations.

Web Services simplify enterprise application integration and create new revenue opportunities by enabling organizations to offer data and services to both customers and partners. Web Services information inquiry has taken a great stride forward because many companies are looking to automate business processes and get products to market faster. The future of Web Services depends on cross-platform interoperability and the creation of a security standard. The Web Services Interoperability Organization will solve these problems.

Service-oriented architecture (SOA) is a good companion to Web Services. It has the benefit of its capacity for rapid modification. It will become an IT architecture mainstream in the future.

Basic Concepts. There are several definitions of Web Services. Here is a typical one: Web Services are self-contained, self-describing business and consumer modular applications, delivered over the Internet, that users can select and combine through almost any device (from personal computers to mobile phones). By using a set of shared protocols and standards, these applications permit different systems to “talk” with one another—that is, to share data and services—without requiring human beings to translate the conversations.

Specifically, a Web Service fits the following three criteria: (1) It is able to expose and describe itself to other applications, allowing those applications to understand what the service does. (2) It can be located by other applications via an online directory, if the service has been registered in a proper directory. (3) It can be invoked by the originating application by using standard protocols.

Web Services have great potential because they can be used in a variety of environments (over the Internet, on an intranet inside a corporate firewall; on an extranet set up by business partners) and can be written using a wide variety of development tools. They can be made to perform a wide variety of tasks, from automating business processes, to integrating components of an enterprisewide system, to streamlining online buying and selling. Key to the promise of Web Services is that, in theory, they can be used by anyone, anywhere, any time, using any hardware and any software, as long as the modular software components of the services are built using a set of key protocols.

The Key Protocols. Web Services are based on a family of key protocols (standards). These protocols are the building blocks of the Web Services platforms. The major protocols are:
XML. Extensible Markup Language makes it easier to exchange data among a variety of applications and to validate and interpret such data. An XML document describes a Web Service and includes information detailing exactly how the Web Service can be run.

SGML. Standard Generalized Markup Language (SGML) is a general standard for the Internet programming languages. It is known informally as “the mother of all Web programming languages.” It sets standards that are independent of any type of computer or of any operating system that sends or retrieves documents. It was developed and standardized by ISO in 1986. It does not specify any formats but rather sets the rules. HTML, XML, and WML are its products.

XML. XML is a WWW Consortium (W3C) standard that translates a company’s business documents into a format understandable by another company. It is the universal format for structured documents and data on the Web. It is intended for open computer-to-computer communications, as it permits the efficient integration of e-commerce solutions across both the Internet and private B2B networks.

XML lets developers define the tags used in terms of the information that tagged elements contain, rather than their appearance. XML code alone will not display anything on the computer screen: only the combination of the HTML code and XML code will serve to display lists and tell the browser what the information is.

According to Microsoft, XML Web Services are the fundamental building blocks in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment in which XML Web Services are becoming the platform for application integration. Applications are constructed using multiple XML Web Services from various sources that work together, regardless of where they reside or how they were implemented. One of the primary advantages of the XML Web Services architecture is that it allows programs written in different languages on different platforms to communicate with each other in a standards-based way.

Industry leaders in accounting, financial reporting, and accounting software are working with firms such as Microsoft and IBM to develop a common XML standard for financial reporting. This major initiative, called Extensible Business Reporting Language (XBRL), is an XML-based financial reporting language that supports the transmission of financial reports in a format that can be processed automatically by computers.

SOAP. Simple Object Access Protocol is a set of rules that facilitate XML exchange between network applications. SOAP defines a common standard that allows different Web Services to interoperate (i.e., it enables communications, such as allowing Visual Basic clients to access Java server). It is a platform-independent specification that defines how messages can be sent between two software systems through the use of XML. These messages typically follow a Request/Response pattern (computer-to-computer).

WSDL. The Web Services Description Language is used to create the XML document that describes tasks performed by Web Services. It actually defines the programmatic interface of the Web Services. Tools such as Visual Studio.Net automate the process of accessing the WSDL, read it, and code the application to reference the specific Web Service.

UDDI. Universal Description, Discovery, and Integration allows for the creation of public or private searchable directories of Web Services. It is the registry of Web Services descriptions. UDDI was developed by the Organization for the Advancement of Structured Information Systems (OASIS), which was formed by IBM, Microsoft, Sun, and others.

Security protocols. Several security standards are in development such as Security Assertion Markup Language (SAML), which is a standard for authentication and authorization. Other security standards are XML signature, XML encryption, XKMS, and XACML.
The Notion of Web Services as Components. Traditionally, people view information systems, including the Web, as relating to information (data) processing. Web Services enable the Web to become a platform for applying business services as components in IT applications. For example, user authentication, currency conversion, and shipping arrangement are components of broad business processes or applications, such as e-commerce ordering or e-procurement systems.

A Web Services Example. As a simple example of how Web Services operate, consider an airline Web site that provides consumers with the opportunity to purchase tickets online. The airline recognizes that customers also might want to rent a car and reserve a hotel as part of their travel plans. The consumer would like the convenience of logging onto only one system rather than three, saving time and effort. Also, the same consumer would like to input personal information only once.

The airline does not have car rental or hotel reservation systems in place. Instead, the airline relies on car rental and hotel partners to provide Web Services access to their systems. The specific services the partners provide are defined by a series of WSDL documents. When a customer makes a reservation for a car or hotel on the airline’s Web site, SOAP messages are sent back and forth in the background between the airline’s and the partners’ servers. In setting up their systems, there is no need for the partners to worry about the hardware or operating systems each is running. Web Services overcome the barriers imposed by these differences.