Object oriented systems analysis and design IS 204

By

Ass.pro. sahera a. saad

2017-2018

Course Objectives:

This module aims to as to introduce variety of new software used by analysts, designers to manage projects, analyze and document systems, design new systems and implement their plans. It introduces also a recent coverage of UML, wireless technologies and ERP; web based systems for e-commerce and expanded coverage on RAD and GUI design.

- Understand the principles and tools of systems analysis and design

Vocabulary: -

Chapter One
Introduction to Systems Analysis and Design

INTRODUCTION

The *systems development life cycle (SDLC)* is the process of understanding how an information system (IS) can support business needs by designing a system, building it, and delivering it to users.
The key person in the SDLC is the systems analyst, who analyzes the business situation, identifies opportunities for improvements, and designs an information system to implement them.

THE SYSTEMS DEVELOPMENT LIFE CYCLE

The SDLC has a set of four fundamental *phases*: planning, analysis, design, and implementation. Each *phase* is itself composed of a series of *steps,* which rely upon *techniques* that produce *deliverables* (specific documents and files that provide understanding about the project).
In many projects, the SDLC phases and steps proceed in a logical path from start to finish . In other projects, the project teams move through the steps consecutively, incrementally, iteratively, or in other patterns.

Planning

The *planning phase* is the fundamental process of understanding *why* an information system should be built and determining how the project team will go about building it. It has two steps:
1. During *project initiation,* the system's business value to the organization is identified: how will it lower costs or increase revenues? Most ideas for new systems

come from outside the IS area (from the marketing department, accounting department, etc.) in the form of a *system request.* A system request presents a brief summary of a business need, and it explains how a system that supports the need will create business value. The IS department works together with the person or department that generated the request (called the *project sponsor*) to conduct a *feasibility analysis*.

The feasibility analysis examines key aspects of the proposed project:

■The idea's technical feasibility (Can we build it?)

■The economic feasibility (Will it provide business value?)

■The organizational feasibility (If we build it, will it be used?)

The system request and feasibility analysis are presented to an information systems *approval committee* (sometimes called a steering committee), which decides whether the project should be undertaken.

2. Once the project is approved, it enters *project management.* During project management, the *project manager* creates a *workplan,* staffs the project, and puts techniques in place to help the project team control and direct the project through the entire SDLC. The deliverable for project management is a *project plan,* which describes how the project team will go about developing the system.

<u>Analysis</u>

The *analysis phase* answers the questions of *who* will use the system, *what* the system will do, and *where* and *when* it will be used. During this phase, the project team investigates any current system(s), identifies improvement opportunities, and develops a concept for the new system.

1. An *analysis strategy* is developed to guide the project team's efforts. Such a strategy usually includes an analysis of the current system (called the *as-is system*) and its problems, and then ways to design a new system (called the *to-be system*).

2. The next step is *requirements gathering* (e.g., through interviews or questionnaires).

The analysis of this information—in conjunction with input from project sponsor and many other people—leads to the development of a concept for a new system. The system concept is then used as a basis to develop a set of business *analysis models,* which describe how the business will operate if the new system is developed. The set of models typically includes models that represent the data and processes necessary to support the underlying business process.

3. The analyses, system concept, and models are combined into a document called the *system proposal,* which is presented to the project sponsor and other key decision makers (e.g., members of the approval committee) who decide whether the project should continue to move forward.

<u>Design</u>

The *design phase* decides *how* the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms and reports; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system were made in the development of the system concept during the analysis phase, the steps in the design phase determine exactly how the system will operate. The design phase has four steps:

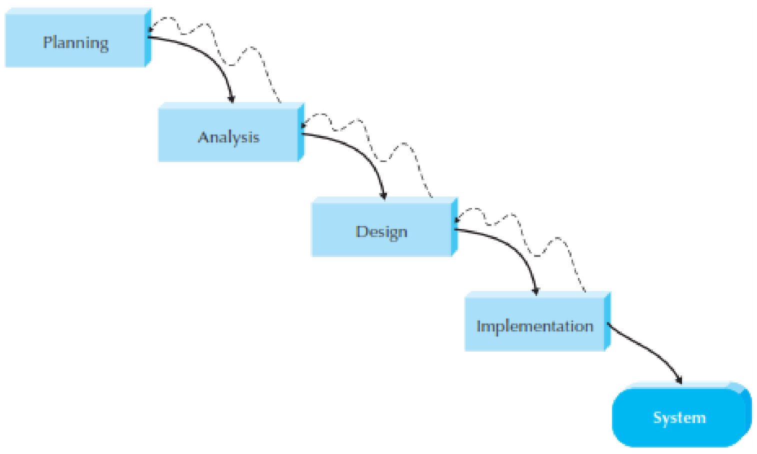
1. The *design strategy* is first developed. It clarifies whether the system will be developed by the company's own programmers, whether the system will be outsourced to another firm (usually a consulting firm), or whether the company will buy an existing software package.
2. This leads to the development of the basic *architecture design* for the system, which describes the hardware, software, and network infrastructure to be used.
3. The *database and file specifications* are developed. These define exactly what data will be stored and where they will be stored.
4. The analyst team develops the *program design,* which defines the programs that need to be written and exactly what each program will do.

<u>Implementation</u>
The final phase in the SDLC is the *implementation phase,* during which the system is actually built (or purchased, in the case of a packaged software design). This is the phase that usually gets the most attention, because for most systems it is the longest and most expensive single part of the development process. This phase has three steps:
1. System *construction* is the first step. The system is built and tested to ensure it performs as designed. Because the cost of bugs can be immense, testing is one of the most critical steps in implementation.
2. The system is installed. *Installation* is the process by which the old system is turned off and the new one is turned on. It may include a direct cutover approach (in which the new system immediately replaces the old system), a parallel conversion approach (in which both the old and new systems are operated for a month or two until it is clear that there are no bugs in the new system), or a phased conversion strategy (in which the new system is installed in one part of the organization as an initial trial and then gradually installed in others).
3. The analyst team establishes a *support plan* for the system.

SYSTEMS DEVELOPMENT METHODOLOGIES
A *methodology* is a formalized approach to implementing the SDLC (i.e., it is a list of steps and deliverables). There are many different systems development methodologies, and each one is unique, based on the order and focus it places on each SDLC phase.

Structured Design
The first category of systems development methodologies is called *structured design.* These methodologies became dominant in the 1980s, replacing the previous, ad hoc, and undisciplined approach.
Structured design methodologies adopt a formal step-by-step approach to the SDLC that moves logically from one phase to the next. Numerous process-centered and data-centered methodologies follow the basic approach of the two structured design categories outlined next.
1.Waterfall Development The original structured design methodology (still used today) is *waterfall development.* With waterfall development–based methodologies, the analysts and users proceed in sequence from one phase to the next (see Figure 1-2). The key deliverables for each phase are typically very long (often hundreds of pages in length) and are presented to the project sponsor for approval as the project moves from phase to phase. Once the sponsor approves the work that was conducted for a phase, the phase ends and the next one begins.

FIGURE 1-2
A Waterfall Development–based Methodology



2.Parallel Development *Parallel development* methodology attempts to address the problem of long delays between the analysis phase and the delivery of the system. Instead of doing design and implementation in sequence, it performs a general design for the whole system and then divides the project into a series of distinct subprojects that can be designed and implemented in parallel. Once all subprojects are complete, there is a final integration of the separate pieces, and the system is delivered (see Figure 1-3).

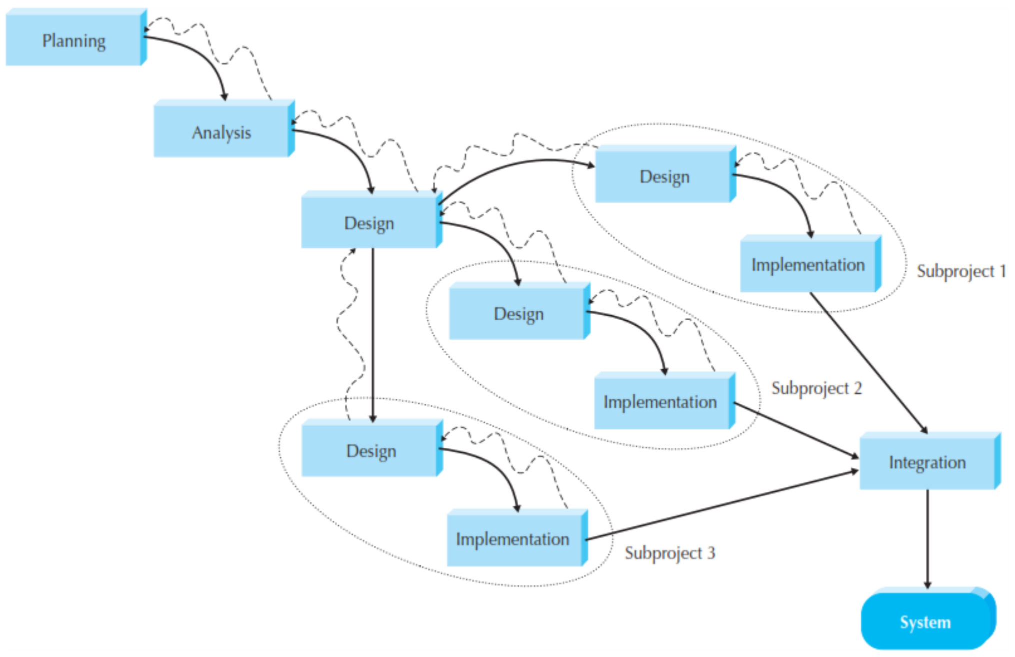


FIGURE 1-3 A Parallel Development–based Methodology

Rapid Application Development (RAD)

A second category of methodologies includes *rapid application development (RAD)–* based methodologies. These are a newer class of systems development methodologies that emerged in the 1990s. RAD-based methodologies attempt to address both weaknesses of structured design methodologies by adjusting the SDLC phases to get some part of the system developed quickly and into the hands of the users.

1. Phased Development A *phased development*–based methodology breaks an overall system into a series of *versions,* which are developed sequentially. The analysis phase identifies the overall system concept, and the project team, users, and system sponsor then categorize the requirements into a series of versions. The most important and fundamental requirements are bundled into the first version of the system. The analysis phase then leads into design and implementation—but only with the set of requirements identified for version 1 (see Figure 1-4).

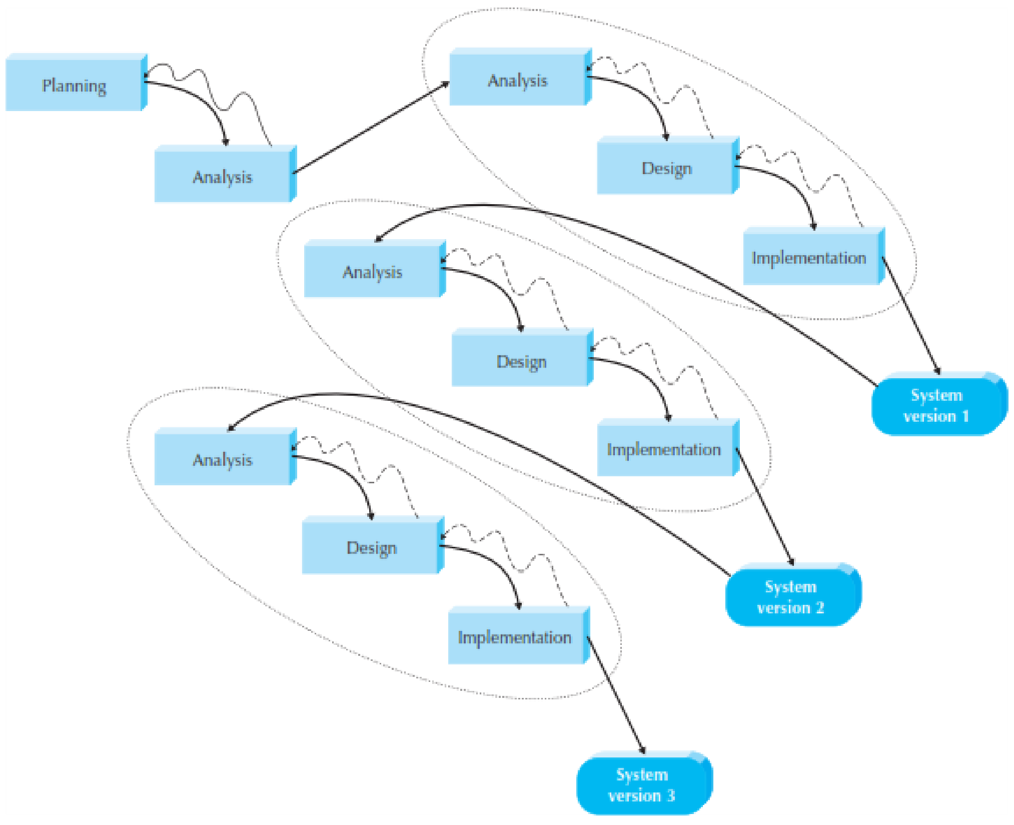


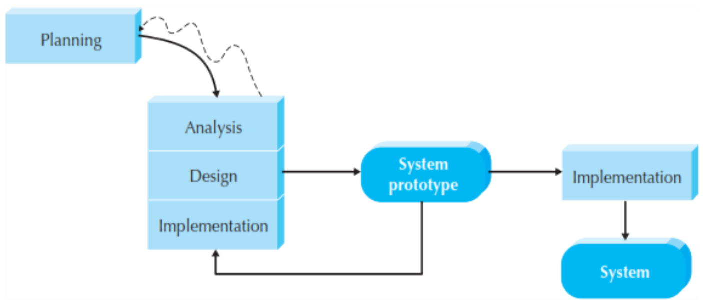
FIGURE 1-4 A Phased Development–based Methodology

Once version 1 is implemented, work begins on version 2. Additional analysis is performed based on the previously identified requirements and combined with new ideas and issues that arose from the users' experience with version 1. Version 2 then is designed and implemented, and work immediately begins on the next version. This process continues until the system is complete or is no longer in use.

2. Prototyping A *prototyping*-based methodology performs the analysis, design, and implementation phases concurrently and all three phases are performed repeatedly in a cycle until the system is completed. With these methodologies, the basics of analysis and design are performed, and work immediately begins on a *system*

prototype, a program that provides a minimal amount of features. The first prototype is usually the first part of the system that is used. This is shown to the users and the project sponsor, who provide comments.
These comments are used to reanalyze, redesign, and reimplement a second prototype, which provides a few more features. This process continues in a cycle until the analysts, users, and sponsor agree that the prototype provides enough functionality to be installed and used in the organization. After the prototype (now called the system) is installed, refinement occurs until it is accepted as the new system (see Figure 1-5).

FIGURE 1-5 A Prototyping-based Methodology



3. Throwaway Prototyping *Throwaway prototyping*–based methodologies are similar to prototyping-based methodologies in that they include the development of prototypes; however, throwaway prototypes are done at a different point in the SDLC. These prototypes are used for a very different purpose than those previously discussed, and they have a very different appearance (see Figure 1-6).

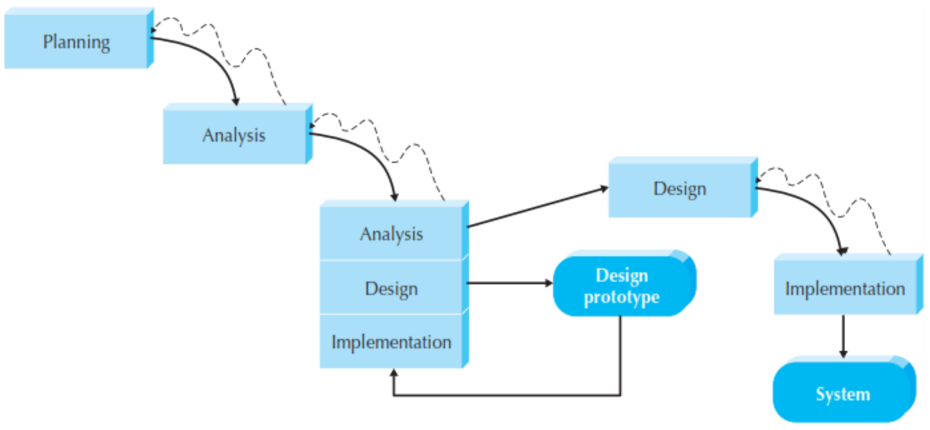


FIGURE 1-6 A Throwaway Prototyping–based Methodology

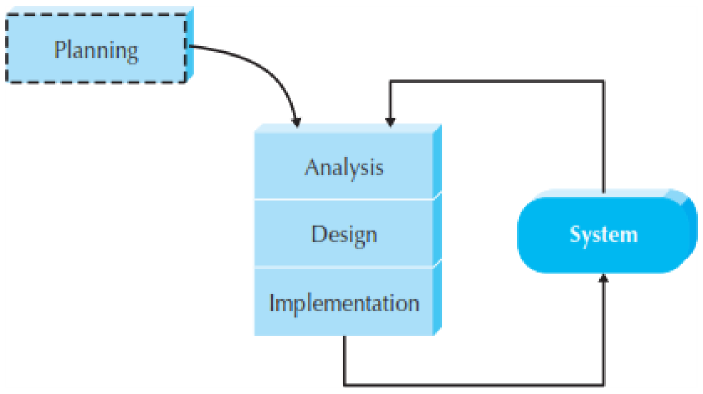
Agile Development
A third category of systems development methodologies is still emerging today: *agile development.* These programming-centric methodologies have few rules and practices, all of which are fairly easy to follow. They focus on streamlining the SDLC by eliminating much of the modeling and documentation overhead and the time spent on those tasks. Instead, projects emphasize simple, iterative application development. Examples of agile development methodologies include extreme programming, Scrum,

and the Dynamic Systems Development Method (DSDM). The agile development approach, as described next, typically is used in conjunction with object-oriented methodologies.

Extreme Programming

Extreme programming (*XP*) is founded on four core values:communication, simplicity, feedback, and courage. These four values provide a foundation that XP developers use to create any system. First, the developers must provide rapid feedback to the end users on a continuous basis. Second, XP requires developers to follow the KISS principle.Third, developers must make incremental changes to grow the system, and they must not only accept change, they must embrace change. Fourth, developers must have a quality-first mentality. XP also supports team members in developing their own skills.Three of the key principles that XP uses to create successful systems are continuous testing, simple coding performed by pairs of developers, and close interactions with end users to build systems very quickly. After a superficial planning process, projects perform analysis, design, and implementation phases iteratively (see Figure 1-7).

FIGURE 1-7
The Extreme Programming Methodology



Clarity of User Requirements When the user requirements for a system are unclear, it is difficult to understand them by talking about them and explaining them with written reports. Users normally need to interact with technology to really understand what a new system can do and how to best apply it to their needs. Prototyping- and throwaway prototyping–based RAD methodologies are usually more appropriate when user requirements are unclear because they provide prototypes for users to interact with early in the SDLC.

Ability to Develop Systems	Structured Methodologies			RAD Methodologies		Agile Methodologies
	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP
With Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent
With Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Poor
That Are Complex	Good	Good	Good	Poor	Excellent	Poor
That Are Reliable	Good	Good	Good	Poor	Excellent	Good
With a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent
With Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Good

FIGURE 1-8 Criteria for Selecting a Methodology

Familiarity with Technology

When the system will use new technology with which the analysts and programmers are not familiar (e.g., the first Web development project with Java), early application of the new technology in the methodology will improve the chance of success. If the system is designed without some familiarity with the base technology, risks increase because the tools might not be capable of doing what is needed. Throwaway prototyping–based methodologies are particularly appropriate if users lack familiarity with technology because they explicitly encourage the developers to develop design prototypes for areas with high risks. Phased development–based methodologies are good as well, because they create opportunities to investigate the technology in some depth before the design is complete. Although you might think prototyping-based methodologies are also appropriate, they are much less so because the early prototypes that are built usually only scratch the surface of the new technology. It is generally only after several prototypes and several months that the developers discover weaknesses or problems in the new technology.

System Complexity

Complex systems require careful and detailed analysis and design. Throwaway prototyping–based methodologies are particularly well suited to such detailed analysis and design, as opposed to prototyping-based methodologies, which are not. The traditional structured design–based methodologies can handle complex systems, but without the ability to get the system or prototypes into the users' hands early on, some key issues may be overlooked. Although phased development–based methodologies enable users to interact with the system early in the process, we have observed that project teams who follow these tend to devote less attention to the analysis of the complete problem domain than they might using other methodologies.

System Reliability

System reliability is usually an important factor in system development after all, who wants an unreliable system? However, reliability is just one factor among several. For some applications reliability is truly critical (e.g., medical equipment, missile control systems), whereas for other applications (e.g., games, Internet video) it is merely important. Throwaway prototyping methodologies are the most appropriate when system reliability is a high priority, because it combines detailed analysis and design phases with the ability for the project team to test many different approaches through design prototypes before completing the design.

Prototyping methodologies are generally not a good choice when reliability is critical because it lacks the careful analysis and design phases that are essential for dependable systems.

Short Time Schedules Projects that have short time schedules are well suited for RAD based methodologies. This is due to them being designed to increase the speed of development. Prototyping and phased development–based methodologies are excellent choices when timelines are short because they best enable the project team to adjust the functionality in the system based on a specific delivery date, and if the project schedule starts to slip, it can be readjusted by removing functionality from the version or prototype under development.
Waterfall-based methodologies are the worst choice when time is at a premium because they do not allow for easy schedule changes.
Schedule Visibility One of the greatest challenges in systems development is determining whether a project is on schedule. This is particularly true of the structured design methodologies because design and implementation occur at the end of the project. The RAD-based methodologies move many of the critical design decisions earlier in the project to help project managers recognize and address risk factors and keep expectations in check.

OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN (OOSAD)

Object-oriented approaches to developing information systems, technically speaking, can use any of the traditional methodologies (waterfall development, parallel development, phased development, prototyping, and throwaway prototyping). However, the object oriented approaches are most associated with a phased development RAD methodology. The primary difference between a traditional approach like structured design and an object oriented approach is how a problem is decomposed. In traditional approaches, the problem decomposition process is either process centric or data centric. However, processes and data are so closely related that it is difficult to pick one or the other as the primary focus. Based on this lack of congruence with the real world, new *object-oriented methodologies* have emerged that use the RAD-based sequence of SDLC phases but attempt to balance the emphasis between process and data by focusing the decomposition of problems on objects that contain both data and processes. Both approaches are valid approaches to developing information systems. In this book, we focus only on object-oriented approaches. According to the creators of the Unified Modeling Language (UML), Grady Booch, Ivar Jacobson, and James Rumbaugh, any modern object-oriented approach to developing information systems must be (1) use-case driven, (2) architecture-centric, and (3) iterative and incremental.

Use-Case Driven

Use-case driven means that *use cases* are the primary modeling tools defining the behavior of the system. A use case describes how the user interacts with the system to perform some activity, such as placing an order, making a reservation, or searching for information.
The use cases are used to identify and to communicate the requirements for the system to the programmers who must write the system.Use cases are inherently simple because they focus on only one activity at a time. In contrast, the process model diagrams used by traditional structured and RAD methodologies are far more Complex because they require the system analyst and user to develop models of the entire system. With traditional methodologies, each business activity is decomposed into a set of subprocesses, which are, in turn, decomposed into further subprocesses, and so on. This goes on until no further process decomposition makes sense, and it

often requires dozens of pages of interlocking diagrams. In contrast, use cases focus on only one activity at a time, so developing models is much simpler.

Architecture Centric

Any modern approach to systems analysis and design should be architecture centric. *Architecture centric* means that the underlying software architecture of the evolving system specification drives the specification, construction, and documentation of the system. Modern object-oriented systems analysis and design approaches should support at least three separate but interrelated architectural views of a system: functional, static, and dynamic. The *functional,* or *external, view* describes the behavior of the system from the perspective of the user. The *structural,* or *static, view* describes the system in terms of attributes, methods, classes, and relationships. The *behavioral,* or *dynamic, view* describes the behavior of the system in terms of messages passed among objects and state changes within an object.

Iterative and Incremental

Modern object-oriented systems analysis and design approaches emphasize *iterative* and *incremental* development that undergoes continuous testing and refinement throughout the life of the project. This implies that the systems analysts develop their understanding of a user's problem by building up the three architectural views little by little.

The systems analyst does this by working with the user to create a functional representation of the system under study. Next, the analyst attempts to build a structural representation of the evolving system. Using the structural representation of the system, the analyst distributes the functionality of the system over the evolving structure to create a behavioral representation of the evolving system.

As an analyst works with the user in developing the three architectural views of the evolving system, the analyst will iterate over each of and among the views. That is, as the analyst better understands the structural and behavioral views, the analyst will uncover missing requirements or misrepresentations in the functional view. This, in turn, can cause changes to be cascaded back through the structural and behavioral views. All three architectural views of the system are interlinked and dependent on each other (see Figure 1-9). As each increment and iteration is completed, a more complete representation of the user's real functional requirements are uncovered.

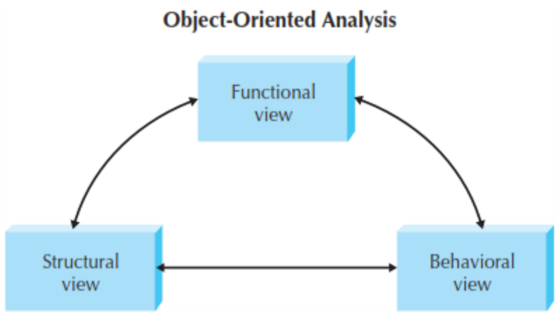


FIGURE 1-9 Iterative and Incremental Development

THE UNIFIED PROCESS

The Unified Process is a specific methodology that maps out when and how to use the various UML techniques for object-oriented analysis and design. The primary contributors were Grady Booch, Ivar Jacobsen, and James Rumbaugh of Rational. Whereas the UML provides structural support for developing the structure and behavior of an information system, the Unified Process provides the behavioral support. The Unified Process, of course, is use-case driven, architecture centric, and iterative and incremental.

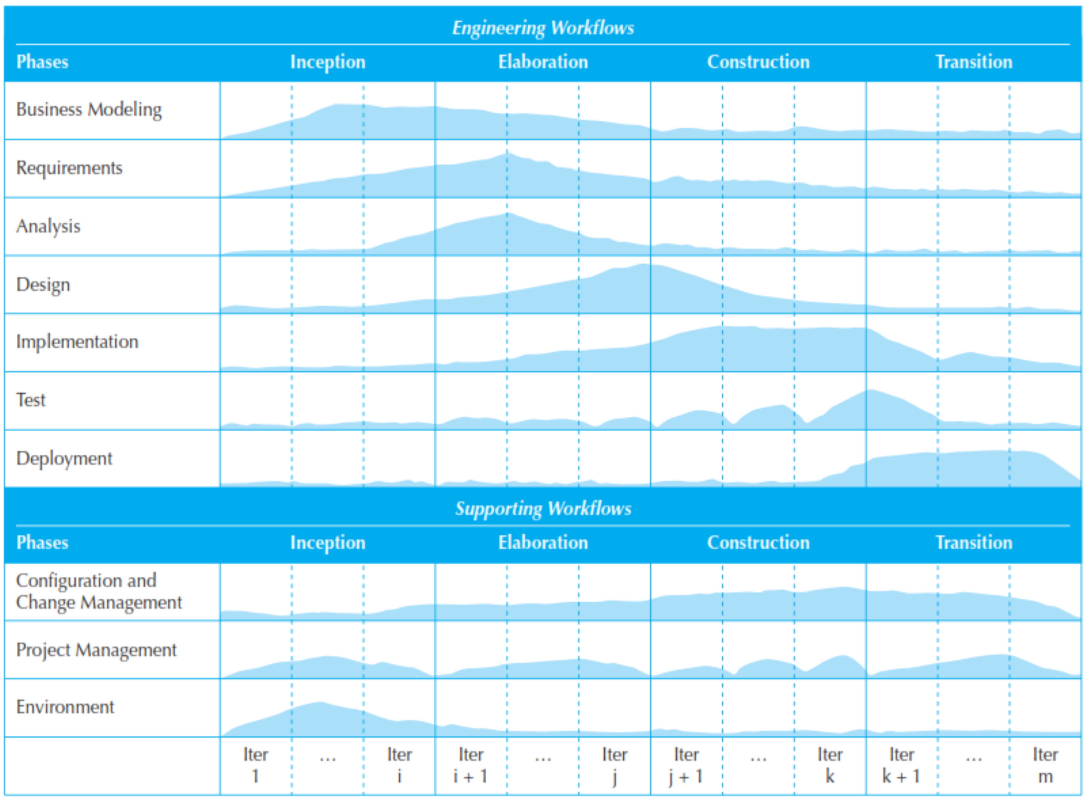


FIGURE 1-10 The Unified Process

The Unified Process is a two-dimensional systems development process described by a set of phases and workflows. The phases are inception, elaboration, construction, and transition. The workflows include business modeling, requirements, analysis, design, implementation, test, deployment, project management, configuration and change management, and environment. In the remainder of this section, we describe the phases and workflows of the Unified Process. Figure 1-10 depicts the Unified Process.

Phases

The *phases* of the Unified Process support an analyst in developing information systems in an iterative and incremental manner. The phases describe how an information system evolves through time. Depending on which development phase the evolving system is currently in, the level of activity will vary over the *workflows.* The curve in Figure 1-10 associated with each workflow approximates the amount of activity that takes place during the specific phase. For example, the inception phase

primarily involves the business modeling and requirements workflows, while practically ignoring the test and deployment workflows.
Each phase contains a set of iterations, and each iteration uses the various workflows to create an incremental version of the evolving information system. As the system evolves through the phases, it improves and becomes more complete. Each phase has objectives, a focus of activity over the workflows, and incremental deliverables. Each of the phases is described next.

Inception In many ways, the *inception phase* is very similar to the planning phase of a traditional SDLC approach. In this phase, a business case is made for the proposed system. This includes feasibility analysis that should answer questions such as the following:

- Do we have the technical capability to build it (technical feasibility)?
- If we build it, will it provide business value (economic feasibility)?
- If we build it, will it be used by the organization (organizational feasibility)?

To answer these questions, the development team performs work related primarily to the business modeling, requirements, and analysis workflows. In some cases, depending on the technical difficulties that could be encountered during the development of the system, a throwaway prototype is developed. This implies that the design, implementation, and test workflows could also be involved. The project management and environment supporting workflows are very relevant to this phase. The primary deliverables from the inception phase are (1) a vision document that sets the scope of the project, identifies the primary requirements and constraints, sets up an initial project plan, and describes the feasibility of and risks associated with the project, and (2) the adoption of the necessary environment to develop the system.

Elaboration When we typically think about object-oriented systems analysis and design, the activities related to the *elaboration phase* of the Unified Process are the most relevant. The *analysis* and *design workflows* are the primary focus during this phase. The elaboration phase continues with developing the vision document, including finalizing the business case, revising the risk assessment, and completing a project plan in sufficient detail to allow the stakeholders to be able to agree with constructing the actual final system. It deals with gathering the requirements, building the UML structural and behavioral models of the problem domain, and detailing how the problem domain models fit into the evolving system architecture.
Developers are involved with all but the *deployment engineering workflow* in this phase. As the developers iterate over the workflows, the importance of addressing configuration and change management becomes apparent. Also, the development tools acquired during the inception phase become critical to the success of the project during this phase.The primary deliverables of this phase include (1) the UML structure and behavior diagrams and (2) an executable of a baseline version of the evolving information system. The baseline version serves as the foundation for all later iterations. By providing a solid foundation at this point in time, the developers have a basis for completing the system in the construction and transition phases.

Construction The *construction phase* focuses heavily on programming the evolving information system. As such, it is primarily concerned with the *implementation workflow*. However, the *requirements workflow* and the analysis and design workflows also are involved with this phase. It is during this phase that missing requirements are uncovered, and the analysis and design models are finally completed. Typically, there are iterations of the workflows during this phase, and during the last iteration, the deployment workflow kicks into high gear. The *configuration and change management workflow*, with its version control activities, becomes extremely important during the construction phase. At times, an iteration may have to be rolled back. Without good version controls, rolling back to a previous version (incremental implementation) of the system is nearly impossible. The primary deliverable of this phase is an implementation of the system that can be released for beta and acceptance testing.

Transition Like the construction phase, the *transition phase* addresses aspects typically associated with the implementation phase of a traditional SDLC approach. Its primary focus is on the testing and deployment workflows. Essentially, the business modeling, requirements, and analysis workflows should have been completed in earlier iterations of the evolving information system.

Depending on the results from the testing workflow, it is possible that some redesign and programming activities on the design and implementation workflows could be necessary, but they should be minimal at this point in time. From a managerial perspective, the project management, configuration and change management, and environment are involved. Some of the activities that take place are beta and acceptance testing, fine-tuning the design and implementation, user training, and the actual rolling out of the final product onto a production platform. Obviously, the primary deliverable is the actual executable information system. The other deliverables include user manuals, a plan to support the users, and a plan for upgrading the information system in the future.

Workflows

The workflows describe the tasks or activities that a developer performs to evolve an information system over time. The workflows of the Unified Process are grouped into two broad categories: engineering and supporting.

Engineering Workflows Engineering workflows include business modeling, requirements, analysis, design, implementation, test, and deployment workflows. The engineering workflows deal with the activities that produce the technical product (i.e., the information system).

Business Modeling Workflow The *business modeling workflow* uncovers problems and identifies potential projects within a user organization. This workflow aids management in understanding the scope of the projects that can improve the efficiency and effectiveness of a user organization. The primary purpose of business modeling is to ensure that both developer and user organizations understand where and how the to-be-developed information system fits into the business processes of the user organization. This workflow is primarily executed during the inception phase to ensure that we develop information systems that make business sense. The activities that take place on this workflow are most closely associated with the planning phase of the traditional SDLC; however, requirements gathering and use-case and business process modeling techniques also help to understand the business situation.

Requirements Workflow In the Unified Process, the requirements workflow includes eliciting both functional and nonfunctional requirements. Typically, requirements are gathered from project stakeholders, such as end users, managers within the end user organization, and even customers. There are many different ways to capture requirements, including interviews, observation techniques, joint application development, document analysis, and questionnaires. The requirements workflow is

utilized the most during the inception and elaboration phases. The identified requirements are very helpful for developing the vision document and the use cases used throughout the development process.
Additional requirements tend to be discovered throughout the development process. In fact, only the transition phase tends to have few, if any, additional requirements identified.
Analysis Workflow The analysis workflow primarily addresses the creation of an analysis model of the problem domain. In the Unified Process, the analyst begins designing the architecture associated with the problem domain; using the UML, the analyst creates structural and behavioral diagrams that depict a description of the problem domain classes and their interactions. The primary purpose of the analysis workflow is to ensure that both the developer and user organizations understand the underlying problem and its domain without overanalyzing.
If they are not careful, analysts can create *analysis paralysis,* which occurs when the project becomes so bogged down with analysis that the system is never actually designed or implemented. A second purpose of the analysis workflow is to identify useful reusable classes for class libraries. By reusing predefined classes, the analyst can avoid "reinventing the wheel" when creating the structural and behavioral diagrams. The analysis workflow is predominantly associated with the elaboration phase, but like the requirements workflow, it is possible that additional analysis will be required throughout the development process.
Design Workflow The design workflow transitions the analysis model into a form that can be used to implement the system: the *design model*. Whereas the analysis workflow concentrated on understanding the problem domain, the design workflow focuses on developing a solution that will execute in a specific environment. Basically, the design workflow simply enhances the description of the evolving information system by adding classes that address the environment of the information system to the evolving analysis model. As such, the design workflow uses activities such as user interface design, database design, physical architecture design, detailed problem domain class design, and the optimization of the including interviews, observation techniques, joint application development, document analysis, and questionnaires.

Implementation Workflow The primary purpose of the implementation workflow is to create an executable solution based on the design model (i.e., programming). This includes not only writing new classes but also incorporating reusable classes from executable class libraries into the evolving solution. As with any programming activity, testing of the new classes and their interactions with the incorporated reusable classes must occur. Finally, in the case of multiple groups performing the implementation of the information system, the implementers also must integrate the separate, individually tested modules to create an executable version of the system. The implementation workflow is associated primarily with the elaboration and construction phases.
Testing Workflow The primary purpose of the *testing workflow* is to increase the quality of the evolving system. As such, testing goes beyond the simple unit testing associated with the implementation workflow. In this case, testing also includes testing the integration of all modules used to implement the system, user acceptance testing, and the actual alpha testing of the software. Practically speaking, testing should go on throughout the development of the system; testing of the analysis and design models occurs during the elaboration and construction phases, whereas

implementation testing is performed primarily during the construction and, to some degree, transition phases. Basically, at the end of each iteration during the development of the information system, some type of test should be performed.

Deployment Workflow The deployment workflow is most associated with the transition phase of the Unified Process. The deployment workflow includes activities, such as software packaging, distribution, installation, and beta testing. When actually deploying the new information system into a user organization, the developers may have to convert the current data, interface the new software with the existing software, and provide end user training on the use of the new system.
Supporting Workflows The supporting workflows include the project management, configuration and change management, and the environment workflows. The supporting workflows focus on the managerial aspects of information system development.
Project Management Workflow Whereas the other workflows associated with the Unified Process are technically active during all four phases, the *project management workflow* is the only truly cross-phase workflow. The development process supports incremental and iterative development, so information systems tend to grow or evolve over time. At the end of each iteration, a new incremental version of the system is ready for delivery. The project management workflow is quite important due to the complexity of the two-dimensional development model of the Unified Process (workflows and phases). This workflow's activities include risk identification and management, scope management, estimating the time to complete each iteration and the entire project, estimating the cost of the individual iteration and the whole project, and tracking the progress being made toward the final version of the evolving information system.
Configuration and Change Management Workflow The primary purpose of the configuration and change management workflow is to keep track of the state of the evolving system. In a nutshell, the evolving information system comprises a set of artifacts, including, for example, diagrams, source code, and executables. During the development process, these artifacts are modified. A substantial amount of work—and, hence, dollars—is involved in the development of the artifacts. As such, the artifacts themselves should be handled as any expensive asset would be handled—access controls must be put into place to safeguard the artifacts from being stolen or destroyed. Furthermore, because the artifacts are modified on a regular, if not continuous, basis, good version control mechanisms should be established. Finally, a good deal of project management information needs to be captured (e.g., author, time, and location of each modification). The configuration and change management workflow is associated mostly with the construction and transition phases.

Environment Workflow During the development of an information system, the development team needs to use different tools and processes. The *environment workflow* addresses these needs. For example, a computer-aided software engineering tool that supports the development of an object-oriented information system via the UML could be required. Other tools necessary include programming environments, project management tools, and configuration management tools. The environment workflow involves acquiring and installing these tools. Even though this workflow can be active during all of the phases of the Unified Process, it should be involved primarily with the inception phase.

Extensions to the Unified Process

As large and as complex as the Unified Process is, many authors have pointed out a set of critical weaknesses. First, the Unified Process does not address staffing, budgeting, or contract management issues. These activities were explicitly left out of the Unified Process. Second, the Unified Process does not address issues relating to maintenance, operations, or support of the product once it has been delivered. As such, it is not a complete software process; it is only a development process. Third, the Unified Process does not address cross- or interproject issues. Considering the importance of reuse in object-oriented systems development and the fact that in many organizations employees work on many different projects at the same time, leaving out interproject issues is a major omission. To address these omissions, Ambler and Constantine suggest the addition of a production phase and two workflows: the operations and support workflow and the infrastructure management workflow (see Figure 1-11).

In addition to these new workflows, the test, deployment, and environment workflows are modified, and the project management and configuration and change management workflows are extended into the production phase.

These extensions are based on alternative object-oriented software processes: the OPEN process and the Object-Oriented Software Process. The new phase, new workflows, and the modifications and extensions to the existing workflows are described next.

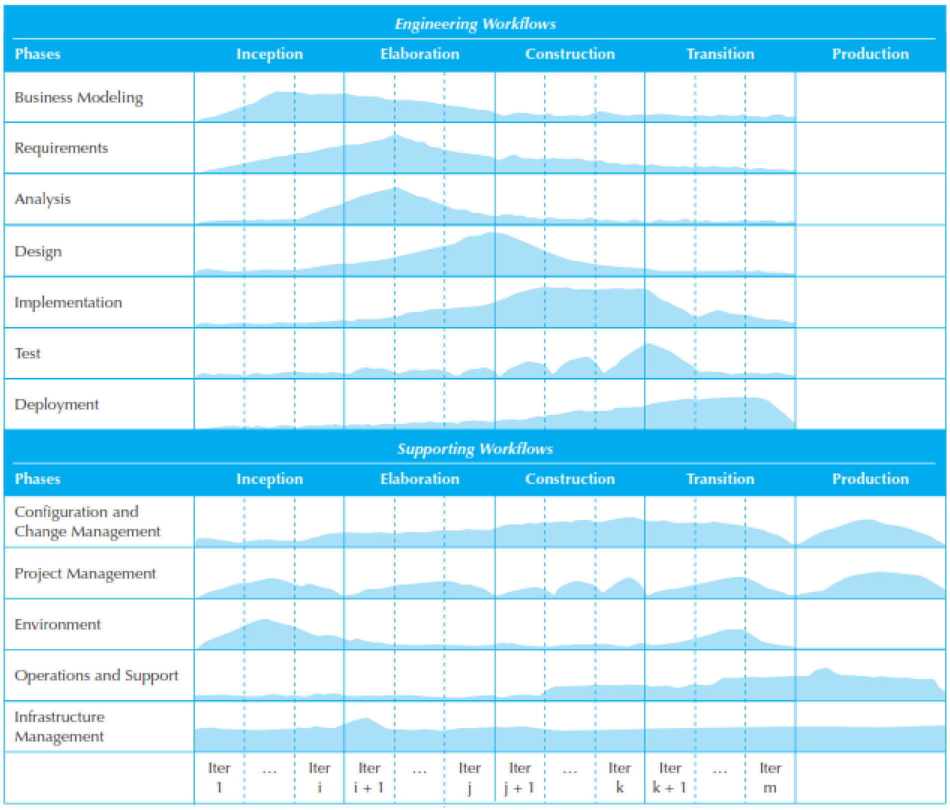


FIGURE 1-11 The Enhanced Unified Process

Production Phase The *production phase* is concerned primarily with issues related to the software product after it has been successfully deployed. This phase focuses on issues related to updating, maintaining, and operating the software. Unlike the

previous phases, there are no iterations or incremental deliverables. If a new release of the software is to be developed, then the developers must begin a new run through the first four phases. Based on the activities that take place during this phase, no engineering workflows are relevant.
The supporting workflows that are active during this phase include the configuration and change management workflow, the project management workflow, the new operations and support workflow, and the infrastructure management workflow.

Operations and Support Workflow The *operations and support workflow*, as you might guess, addresses issues related to supporting the current version of the software and operating the software on a daily basis. Activities include creating plans for the operation and support of the software product once it has been deployed, creating training and user documentation, putting into place necessary backup procedures, monitoring and optimizing the performance of the software, and performing corrective maintenance on the software. This workflow becomes active during the construction phase; its level of activity increases throughout the transition and, finally, the production phase. The workflow finally drops off when the current version of the software is replaced by a new version. Many developers are under the false impression that once the software has been delivered to the customer, their work is finished. In most cases, the work of supporting the software product is much more costly and time consuming than the original development. As such, the developer's work may have just begun.
Infrastructure Management Workflow The *infrastructure management workflow's* primary purpose is to support the development of the infrastructure necessary to develop object-oriented systems. Activities such as development and modification of libraries, standards, and enterprise models are very important. When the development and maintenance of a problem domain architecture model goes beyond the scope of a single project and reuse is going to occur, the infrastructure management workflow is essential. Another very important set of cross-project activities is the improvement of the software development process. Because the activities on this workflow tend to affect many projects and the Unified Process focuses only on a specific project, the Unified Process tends to ignore these activities (i.e., they are simply beyond the scope and purpose of the Unified Process).

Existing Workflow Modifications and Extensions In addition to the workflows that were added to address deficiencies contained in the Unified Process, existing workflows had to be modified and/or extended into the production phase. These workflows include the test, deployment, environment, project management, and configuration and change management workflows.
Test Workflow For high-quality information systems to be developed, testing should be done on every deliverable, including those created during the inception phase. Otherwise, less than quality systems will be delivered to the customer.
Deployment Workflow Legacy systems exist in most corporations today, and these systems have databases associated with them that must be converted to interact with the new systems.
Due to the complexity of deploying new systems, the conversion requires significant planning. As such, the activities on the deployment workflow need to begin in the inception phase instead of waiting until the end of the construction phase, as suggested by the Unified Process.
Environment Workflow The environment workflow needed to be modified to include

activities related to setting up the operations and production environment. The actual work performed is similar to the work related to setting up the development environment that was performed during the inception phase. In this case, the additional work is performed during the transition phase.
Project Management Workflow Even though the project management workflow does not include staffing the project, managing the contracts among the customers and vendors, and managing the project's budget, these activities are crucial to the success of any software development project. As such, we suggest extending project management to include these activities. Furthermore, this workflow should additionally occur in the production phase to address issues such as training, staff management, and client relationship management.
Configuration and Change Management Workflow The configuration and change management workflow is extended into the new production phase. Activities performed during the production phase include identifying potential improvements to the operational system and assessing the potential impact of the proposed changes. Once developers have identified these changes and understood their impact, they can schedule the changes to be made and deployed with future releases.
Figure 1-12 shows the chapters in which the Enhanced Unified Process's phases and workflows are covered. Given the offshore outsourcing and automation of information technology,in this textbook, we focus primarily on the elaboration phase and the business modeling, requirements, analysis, design, and project management workflows of the Enhanced Unified Process. However, as Figure 1-12 shows, the other phases and workflows are covered. In many object-oriented systems development environments today, code generation is supported. Thus, from a business perspective, we believe the activities associated with these workflows are the most important.

THE UNIFIED MODELING LANGUAGE

Until 1995, object concepts were popular but implemented in many different ways by different developers. Each developer had his or her own methodology and notation (e.g., Booch, Coad, Moses, OMT, OOSE, SOMA.)Then in 1995, Rational Software brought three industry leaders together to create a single approach to object-oriented systems development. Grady Booch, Ivar Jacobson, and James Rumbaugh worked with others to create a standard set of diagramming techniques known as the *Unified Modeling Language (UML)*. The objective of UML was to provide a common vocabulary of object-oriented terms and diagramming techniques rich enough to model any systems development project from analysis through implementation. In November 1997, the *Object Management Group (OMG)* formally accepted UML as the standard for all object developers. During the following years, the UML has gone through multiple minor revisions. The current version of UML, Version 2.0, was accepted by the members of the OMG during their spring and summer meetings of 2003.Version 2.0 of the UML defines a set of fourteen diagramming techniques used to model a system. The diagrams are broken into two major groupings: one for modeling structure of a system and one for modeling behavior. *Structure diagrams* provide a way to represent the data and static relationships in an information system.

The structure diagrams include class, object, package, deployment, component, and composite structure diagrams.
Behavior diagrams provide the analyst with a way to depict the dynamic relationships among the instances or objects that represent the business information system. They also allow the modeling of the dynamic behavior of individual objects throughout their lifetime. The behavior diagrams support the analyst in modeling the functional requirements of an evolving information system. The behavior modeling diagrams include activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine,and use-case diagrams.Figure 1-13 provides an overview of these diagrams.
Depending on where in the development process the system is, different diagrams play a more important role. In some cases, the same diagramming technique is used throughout the development process. In that case, the diagrams start off very conceptual and abstract. As the system is developed, the diagrams evolve to include details that ultimately lead to code generation and development. In other words, the diagrams move from documenting the requirements to laying out the design. Overall, the consistent notation, integration among the diagramming techniques, and application of the diagrams across the entire development process makes the UML a powerful and flexible language for analysts and developers. Later chapters provide more detail on using a subset of the UML in object-oriented systems analysis and design. In particular, these chapters describe activity, use-case, class, object, sequence, communication, and package diagrams and the behavioral state machines.

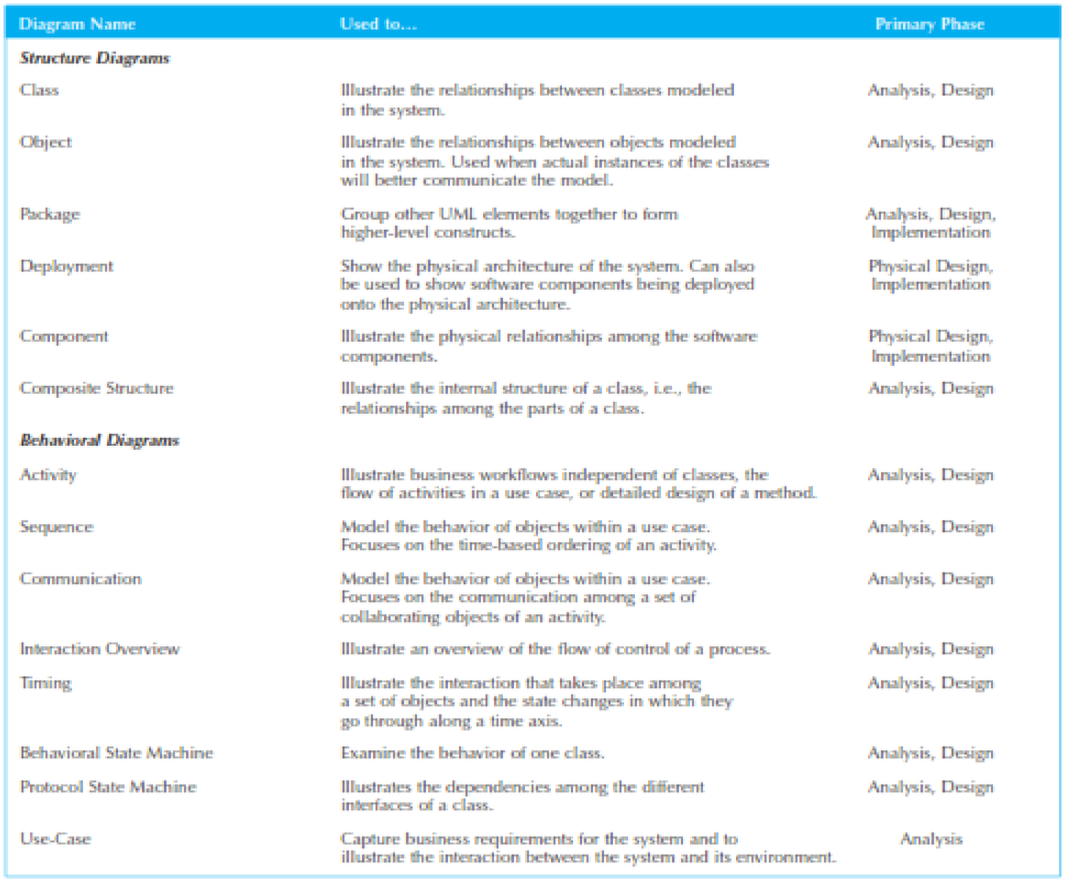


Diagram Name	Used to...	Primary Phase
Structure Diagrams		
Class	Illustrate the relationships between classes modeled in the system.	Analysis, Design
Object	Illustrate the relationships between objects modeled in the system. Used when actual instances of the classes will better communicate the model.	Analysis, Design
Package	Group other UML elements together to form higher-level constructs.	Analysis, Design, Implementation
Deployment	Show the physical architecture of the system. Can also be used to show software components being deployed onto the physical architecture.	Physical Design, Implementation
Component	Illustrate the physical relationships among the software components.	Physical Design, Implementation
Composite Structure	Illustrate the internal structure of a class, i.e., the relationships among the parts of a class.	Analysis, Design
Behavioral Diagrams		
Activity	Illustrate business workflows independent of classes, the flow of activities in a use case, or detailed design of a method.	Analysis, Design
Sequence	Model the behavior of objects within a use case. Focuses on the time-based ordering of an activity.	Analysis, Design
Communication	Model the behavior of objects within a use case. Focuses on the communication among a set of collaborating objects of an activity.	Analysis, Design
Interaction Overview	Illustrate an overview of the flow of control of a process.	Analysis, Design
Timing	Illustrate the interaction that takes place among a set of objects and the state changes in which they go through along a time axis.	Analysis, Design
Behavioral State Machine	Examine the behavior of one class.	Analysis, Design
Protocol State Machine	Illustrates the dependencies among the different interfaces of a class.	Analysis, Design
Use-Case	Capture business requirements for the system and to illustrate the interaction between the system and its environment.	Analysis

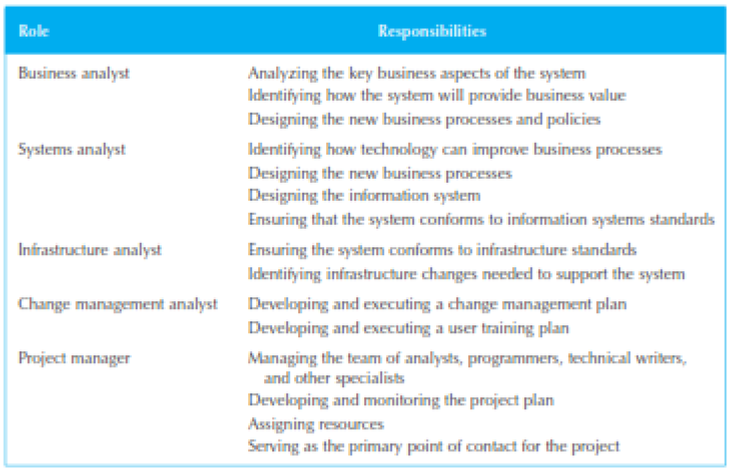
FIGURE 1-13 UML 2.0 Diagram Summary

PROJECT TEAM ROLES AND SKILLS

It is clear from the various phases and steps performed during the SDLC that the project team needs a variety of skills. Project members are *change agents* who identify ways to improve an organization, build an information system to support them, and train and motivate others to use the system. Leading a successful organizational change effort is one of the most difficult jobs that someone can do. Understanding what to change and how to change it—and convincing others of the need for change—requires a wide range of skills. These skills can be broken down into six major categories: technical, business, analytical, interpersonal, management, and ethical.

Analysts must have the technical skills to understand the organization's existing technical environment, the technology that will comprise the new system, and the way in which both can be fit into an integrated technical solution. Business skills are required to understand how IT can be applied to business situations and to ensure that the IT delivers real business value. Analysts are continuous problem solvers at both the project and the organizational level, and they put their analytical skills to the test regularly. Analysts often need to communicate effectively one-on-one with users and business managers (who often have little experience with technology) and with programmers (who often have more technical expertise than the analyst). They must be able to give presentations to large and small groups and write reports. Not only do they need to have strong interpersonal abilities, but they also need to manage people with whom they work and they need to manage the pressure and risks associated with unclear situations.

Finally, analysts must deal fairly, honestly, and ethically with other project team members, managers, and system users. Analysts often deal with confidential information or information that, if shared with others, could cause harm (e.g., dissent among employees); it is important to maintain confidence and trust with all people. In addition to these six general skill sets, analysts require many specific skills associated with roles performed on a project. In the early days of systems development, most organizations expected one person, the analyst, to have all the specific skills needed to conduct a systems development project. Some small organizations still expect one person to perform many roles, but because organizations and technology have become more complex, most large organizations now build project teams containing several individuals with clearly defined responsibilities. Different organizations divide the roles differently, but Figure 1-14 presents one commonly used set of project team roles. Most IS teams include many other individuals, such as the *programmers,* who actually write the programs that make up the system, and *technical writers,* who prepare the help screens and other documentation (e.g., users manuals and systems manuals).



Role	Responsibilities
Business analyst	Analyzing the key business aspects of the system Identifying how the system will provide business value Designing the new business processes and policies
Systems analyst	Identifying how technology can improve business processes Designing the new business processes Designing the information system Ensuring that the system conforms to information systems standards
Infrastructure analyst	Ensuring the system conforms to infrastructure standards Identifying infrastructure changes needed to support the system
Change management analyst	Developing and executing a change management plan Developing and executing a user training plan
Project manager	Managing the team of analysts, programmers, technical writers, and other specialists Developing and monitoring the project plan Assigning resources Serving as the primary point of contact for the project

FIGURE 1-14
Project Team Roles

Business Analyst

A *business analyst* focuses on the business issues surrounding the system. These issues include identifying the business value that the system will create, developing ideas and suggestions for how the business processes can be improved, and designing the new processes and policies in conjunction with the systems analyst. This individual will likely have business experience and some type of professional training (e.g., the business analyst for accounting systems will likely be a CPA [in the United States] or a CA [in Canada]). He or she represents the interests of the project sponsor and the ultimate users of the system. A business analyst assists in the planning and design phases but is most active in the analysis phase.

Systems Analyst

A *systems analyst* focuses on the IS issues surrounding the system. This person develops ideas and suggestions for how information technology can improve business processes,designs the new business processes with help from the business analyst, designs the new information system, and ensures that all IS standards are maintained. A systems analyst will likely have significant training and experience in analysis and design, programming, and even areas of the business. He or she represents the interests of the IS department and works intensively through the project but perhaps less so during the implementation phase.

Infrastructure Analyst

An *infrastructure analyst* focuses on the technical issues surrounding how the system will interact with the organization's technical infrastructure (e.g., hardware, software, networks, and databases). An infrastructure analyst's tasks include ensuring that the new information system conforms to organizational standards and identifying infrastructure changes needed to support the system. This individual will probably have significant training and experience in networking, database administration, and various hardware and software products. He or she represents the interests of the organization and IS group that will ultimately have to operate and support the new system once it has been installed. An infrastructure analyst works throughout the project but perhaps less so during planning and analysis phases.

Change Management Analyst

A *change management analyst* focuses on the people and management issues surrounding the system installation. The roles of this person include ensuring that the adequate documentation and support are available to users, providing user training on the new system, and developing strategies to overcome resistance to change. This

individual should have significant training and experience in organizational behavior in general and change management in particular.
He or she represents the interests of the project sponsor and users for whom the system is being designed. A change management analyst works most actively during the implementation phase but begins laying the groundwork for change during the analysis and design phases.

Project Manager

A project manager is responsible for ensuring that the project is completed on time and within budget and that the system delivers all benefits intended by the project sponsor. The role of the project manager includes managing the team members, developing the project plan, assigning resources, and being the primary point of contact when people outside the team have questions about the project. This individual will likely have significant experience in project management and has probably worked for many years as a systems analyst beforehand. He or she represents the interests of the IS department and the project sponsor. The project manager works intensely during all phases of the project.

APPLYING THE CONCEPTS AT CD SELECTIONS

Throughout this book, many new concepts about object-oriented systems analysis and design are introduced. As a way to make these new concepts more relevant, we apply them to a fictitious company called CD Selections. CD Selections is a chain of fifty music stores located in California, with headquarters in Los Angeles. Annual sales last year were $50 million, and they have been growing at about 3 to 5 percent per year for the past few years. However, the firm has been interested in expanding their presence beyond California. Margaret Mooney, vice president of marketing, has recently become both excited by and concerned with the rise of Internet sites selling CDs. She believes that the Internet has great potential, but she wants to use it in the right way. Rushing into e-commerce without considering things such as its effect on existing brick-and-mortar stores and the implications on existing systems at CD Selections could cause more harm than good. Currently, CD Selections has a Web site that provides basic information about the company and about each of its stores (e.g., map, operating hours, phone number, etc.). The Web site was developed by an Internet consulting firm and is hosted by a prominent local Internet service provider (ISP) in Los Angeles. The IT department at CD Selections has become experienced with Internet technology as it has worked with the ISP to maintain the site; however, it still has a lot to learn when it comes to conducting business over the Web. As such, Margaret is interested in investigating the possibility of creating an e-commerce site that will work with the current systems used by CD Selections. In future chapters, we revisit CD Selections to see how the concepts introduced in the individual chapters impact Margaret and CD Selections.

SUMMARY

The Systems Development Life Cycle

All systems development projects follow essentially the same fundamental process, called the system development life cycle (SDLC). SDLC starts with a planning phase in which the project team identifies the business value of the system, conducts a feasibility analysis, and plans the project. The second phase is the analysis phase, in which the team develops an analysis strategy, gathers information, and builds a set of analysis models. In the next phase, the design phase, the team develops the physical design, architecture design, interface design, database and file specifications, and program design. In the final phase, implementation, the system is built, installed, and maintained.

The Evolution of Systems Development Methodologies

System development methodologies are formalized approaches to implementing an SDLC. System development methodologies have evolved over the decades. Structured design methodologies, such as waterfall and parallel development, emphasize decomposition of a problem by either focusing on process decomposition (process-centric methodologies) or data decomposition (data decomposition). They produce a solid, well-thought-out system but can overlook requirements because users must specify them early in the design process before seeing the actual system. RAD-based methodologies attempt to speed up development and make it easier for users to specify requirements by having parts of the system developed sooner either by producing different versions (phased development) or by using prototypes (prototyping, throwaway prototyping) through the use of CASE tools and fourth-generation/visual programming languages. However, RAD-based methodologies still tend to be either process-centric or data-centric. Agile development methodologies, such as XP, focus on streamlining the SDLC by eliminating many of the tasks and time associated with requirements definition and documentation. Several factors influence the choice of a methodology: clarity of the user requirements; familiarity with the base technology; system complexity; need for system reliability; time pressures; and the need to see progress on the time schedule.

Object-Oriented Systems Analysis and Design

Object-oriented systems analysis and design (OOSAD) is most associated with a phaseddevelopment RAD-based methodology, where the time spent in each phase is very short. OOSAD uses a use-case-driven, architecture-centric, iterative, and incremental information systems development approach. It supports three different views of the evolving system: functional, static, and dynamic. OOSAD allows the analyst to decompose complex problems into smaller, more manageable components using a commonly accepted set of notations. Also, many people believe that users do not think in terms of data or processes but instead think in terms of a collection of collaborating objects. As such, object-oriented systems analysis and design allows the analyst to interact with the user with objects from the user's environment instead of a set of separate processes and data.

One of the most popular approaches to object-oriented systems analysis and design is the Unified Process. The Unified Process is a two-dimensional systems development process described with a set of phases and workflows. The phases consist of the inception, elaboration, construction, and transition phases. The workflows are organized into two subcategories: engineering and supporting. The engineering workflows include business modeling, requirements, analysis, design, implementation, test, and deployment workflows, and the supporting workflows comprise the project management, configuration and change management, and environment workflows. Depending on which development phase the evolving system is currently in, the level of activity will vary over the workflows.

The Unified Modeling Language

The Unified Modeling Language, or UML, is a standard set of diagramming techniques that provide a graphical representation rich enough to model any systems development project from analysis through implementation. Today most object-oriented systems analysis and design approaches use the UML to depict an evolving system. The UML uses a set of different diagrams to portray the various views of the evolving system. The diagrams are grouped into two broad classifications: structure and behavior. The structure diagrams include class, object, package, deployment, component, and composite structure diagrams. The behavior diagrams include activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine, and use case diagrams.

Project Team Roles and Skills

The project team needs a variety of skills. All analysts need to have general skills, such as change management, ethics, communications, and technical. However, different kinds of analysts require specific skills in addition to these. Business analysts usually have business skills that help them to understand the business issues surrounding the system, whereas systems analysts also have significant experience in analysis and design and programming. The infrastructure analyst focuses on technical issues surrounding how the system will interact with the organization’s technical infrastructure, and the change management analyst focuses on people and management issues surrounding the system installation. In addition to analysts, project teams will include a project manager, programmers, technical writers, and other specialists.

Chapter 2: Project Initiation

INTRODUCTION

The first step in any new development project is for someone—a manager, staff member, sales representative, or systems analyst—to see an opportunity to improve the business. New systems start first and foremost from a business need or opportunity. Many ideas for new systems or improvements to existing ones arise from the application of a new technology, but an understanding of technology is usually secondary to a solid understanding of the business and its objectives.

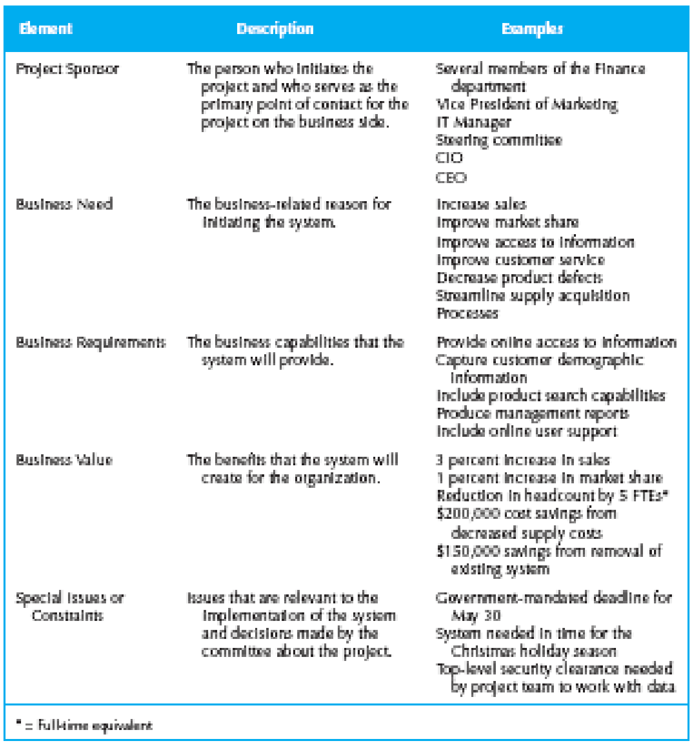
PROJECT IDENTIFICATION

A project is identified when someone in the organization identifies a *business need* to build a system. This could occur within a business unit or IT, come from a steering committee charged with identifying business opportunities, or evolve from a recommendation made by external consultants. Examples of business needs include supporting a new marketing campaign, reaching out to a new type of customer, or improving interactions with suppliers. Sometimes, needs arise from some kind of "pain"within the organization, such as a drop in market share, poor customer service levels, or increased competition.

Other times, new business initiatives and strategies are created, and a system is required to enable them. The project sponsor is someone who recognizes the strong business need for a system and has an interest in seeing the system succeed. He or she will work throughout the SDLC to make sure that the project is moving in the right direction from the perspective of the business. The project sponsor serves as the primary point of contact for the system.

Usually, the sponsor of the project is from a business function, such as marketing, accounting, or finance; however, members of the IT area also can sponsor or cosponsor a project The project sponsor also should have an idea of the *business value* to be gained from the system, both in tangible and intangible ways. *Tangible value* can be quantified and measured easily (e.g., 2 percent reduction in operating costs). An *intangible value* results from an intuitive belief that the system provides important, but hard-to-measure, benefits to the organization (e.g., improved customer service or a better competitive position).

Once the project sponsor identifies a project that meets an important business need and he or she can identify the system's business requirements and value, it is time to formallyinitiate the project. In most organizations, project initiation begins with a technique called a system request.



Element	Description	Examples
Project Sponsor	The person who initiates the project and who serves as the primary point of contact for the project on the business side.	Several members of the Finance department Vice President of Marketing IT Manager Steering committee CIO CEO
Business Need	The business-related reason for initiating the system.	Increase sales Improve market share Improve access to information Improve customer service Decrease product defects Streamline supply acquisition Processes
Business Requirements	The business capabilities that the system will provide.	Provide online access to information Capture customer demographic information Include product search capabilities Produce management reports Include online user support
Business Value	The benefits that the system will create for the organization.	3 percent increase in sales 1 percent increase in market share Reduction in headcount by 5 FTEs* $200,000 cost savings from decreased supply costs $150,000 savings from removal of existing system
Special Issues or Constraints	Issues that are relevant to the implementation of the system and decisions made by the committee about the project.	Government-mandated deadline for May 30 System needed in time for the Christmas holiday season Top-level security clearance needed by project team to work with data

* = Full-time equivalent

FIGURE 2-1
Elements of the System Request Form

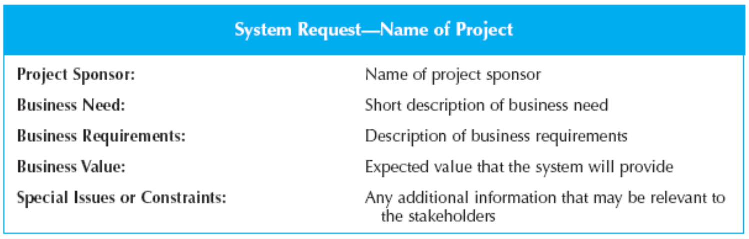
System Request

A system request is a document that describes the business reasons for building a system and the value that the system is expected to provide. The project sponsor usually completes this form as part of a formal system project selection process within the organization.

Most system requests include five elements: project sponsor, business need, business requirements, business value, and *special issues* (see Figure 2-1). The sponsor describes the person who will serve as the primary contact for the project, and the business need presents the reasons prompting the project. The business requirements of the project refer to the business capabilities that the system will need to have, and the business value describes the benefits that the organization should expect from the system.

Special issues are included on the document as a catch-all for other information that should be considered in assessing the project. For example, the project may need to be completed by a specific deadline. Project teams need to be aware of any special circumstances that could affect the outcome of the system. Figure 2-2 shows a template for a system request.

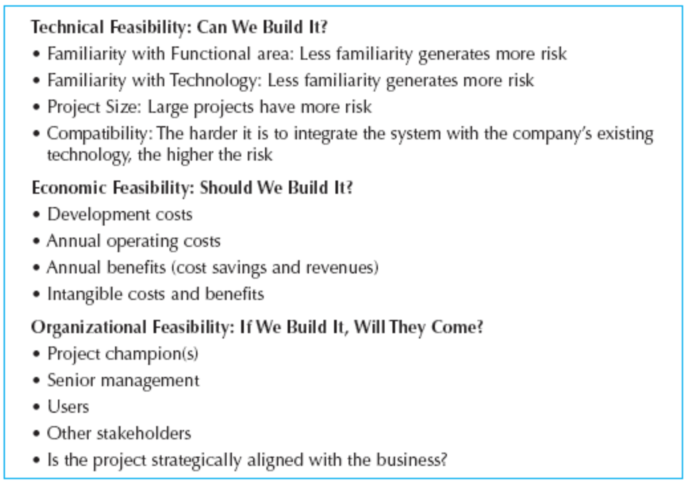
The completed system request is submitted to the approval committee for consideration. This approval committee could be a company steering committee that meets regularly to make information systems decisions, a senior executive who has control of organizational resources, or any other decision-making body that governs the use of business investments. The committee reviews the system request and makes an initial determination, based on the information provided, of whether to investigate the proposal or not. If so, the next step is to conduct a feasibility analysis.



System Request—Name of Project	
Project Sponsor:	Name of project sponsor
Business Need:	Short description of business need
Business Requirements:	Description of business requirements
Business Value:	Expected value that the system will provide
Special Issues or Constraints:	Any additional information that may be relevant to the stakeholders

FIGURE 2-2
System Request Template

Feasibility Analysis
Feasibility analysis guides the organization in determining whether or not to proceed with a project. Feasibility analysis also identifies the important *risks* associated with the project that must be addressed if the project is approved.
As with the system request, each organization has its own process and format for the feasibility analysis, but most include three techniques: technical feasibility, economic feasibility, and organizational feasibility. The results of these techniques are combined into a *feasibility study* deliverable, which is given to the approval committee at the end of project initiation (see Figure 2-3).



Technical Feasibility: Can We Build It?
- Familiarity with Functional area: Less familiarity generates more risk
- Familiarity with Technology: Less familiarity generates more risk
- Project Size: Large projects have more risk
- Compatibility: The harder it is to integrate the system with the company's existing technology, the higher the risk

Economic Feasibility: Should We Build It?
- Development costs
- Annual operating costs
- Annual benefits (cost savings and revenues)
- Intangible costs and benefits

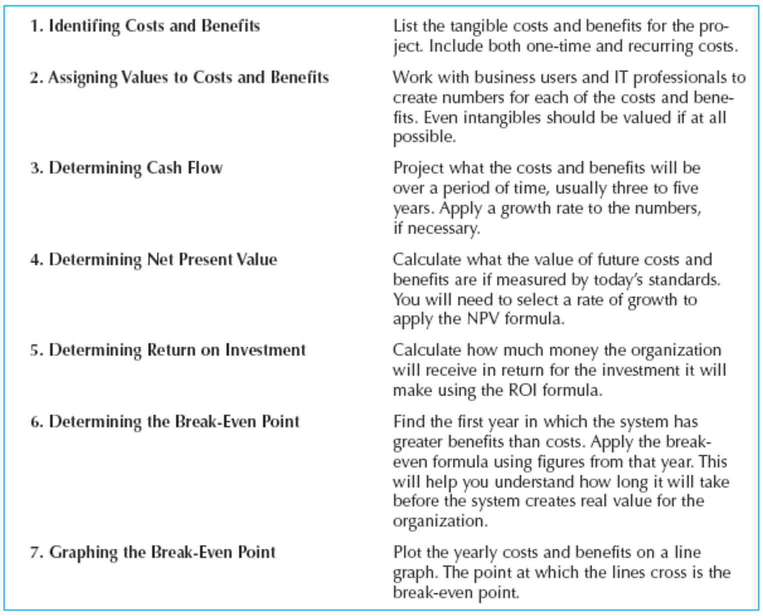
Organizational Feasibility: If We Build It, Will They Come?
- Project champion(s)
- Senior management
- Users
- Other stakeholders
- Is the project strategically aligned with the business?

FIGURE 2-3
Feasibility Analysis Assessment Factors

Technical Feasibility
The first technique in the feasibility analysis is to assess the *technical feasibility* of the project, the extent to which the system can be successfully designed, developed, and installed by the IT group. Technical feasibility analysis is in essence a *technical risk analysis* that strives to answer this question: *Can* we build it?

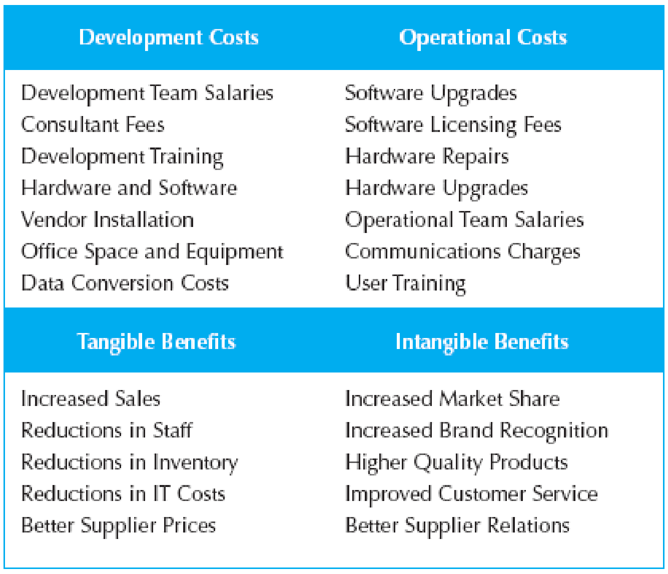
Economic Feasibility
The second element of a feasibility analysis is to perform an *economic feasibility* analysis (also called a *cost–benefit analysis*), which identifies the financial risk associated with the project. It attempts to answer this question: *Should* we build the system? Economic feasibility is determined by identifying costs and benefits associated with the system, assigning values to them, and then calculating the cash flow and return on investment for the project.

The more expensive the project, the more rigorous and detailed the analysis should be. Figure 2-4 lists the steps in performing a cost benefit analysis;



1. Identifing Costs and Benefits	List the tangible costs and benefits for the project. Include both one-time and recurring costs.
2. Assigning Values to Costs and Benefits	Work with business users and IT professionals to create numbers for each of the costs and benefits. Even intangibles should be valued if at all possible.
3. Determining Cash Flow	Project what the costs and benefits will be over a period of time, usually three to five years. Apply a growth rate to the numbers, if necessary.
4. Determining Net Present Value	Calculate what the value of future costs and benefits are if measured by today's standards. You will need to select a rate of growth to apply the NPV formula.
5. Determining Return on Investment	Calculate how much money the organization will receive in return for the investment it will make using the ROI formula.
6. Determining the Break-Even Point	Find the first year in which the system has greater benefits than costs. Apply the break-even formula using figures from that year. This will help you understand how long it will take before the system creates real value for the organization.
7. Graphing the Break-Even Point	Plot the yearly costs and benefits on a line graph. The point at which the lines cross is the break-even point.

FIGURE 2-4
Steps for Conducting Economic Feasibility

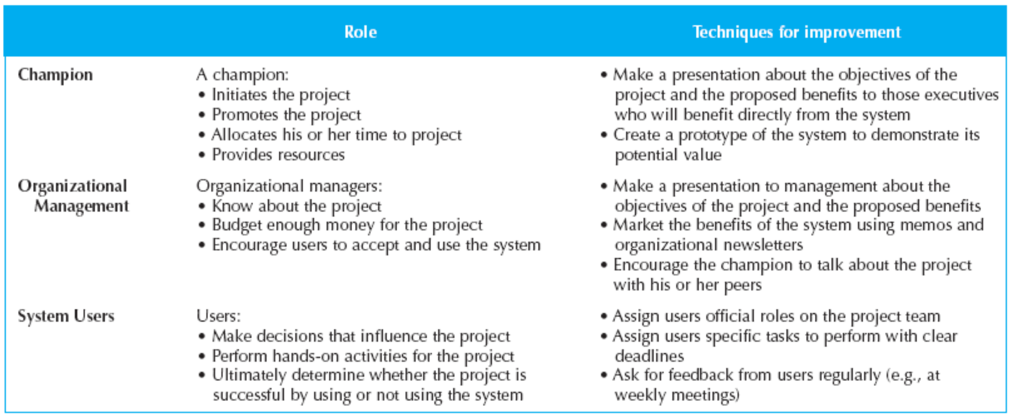


Development Costs	Operational Costs
Development Team Salaries	Software Upgrades
Consultant Fees	Software Licensing Fees
Development Training	Hardware Repairs
Hardware and Software	Hardware Upgrades
Vendor Installation	Operational Team Salaries
Office Space and Equipment	Communications Charges
Data Conversion Costs	User Training
Tangible Benefits	**Intangible Benefits**
Increased Sales	Increased Market Share
Reductions in Staff	Increased Brand Recognition
Reductions in Inventory	Higher Quality Products
Reductions in IT Costs	Improved Customer Service
Better Supplier Prices	Better Supplier Relations

FIGURE 2-5
Example Costs and Benefits for Economic Feasibility

Organizational Feasibility

The final technique used for feasibility analysis is to assess the *organizational feasibility* of the system, how well the system ultimately will be accepted by its users and incorporated into the ongoing operations of the organization. There are many organizational factors that can have an impact on the project, and seasoned developers know that organizational feasibility can be the most difficult feasibility dimension to assess. In essence, an organizational feasibility analysis attempts to answer this question: If we build it, will they come?

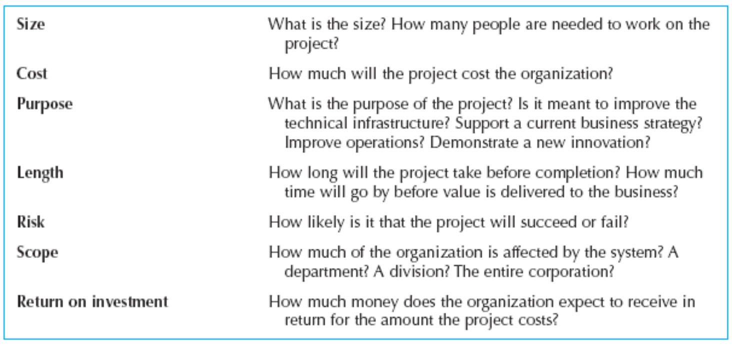


	Role	Techniques for improvement
Champion	A champion: • Initiates the project • Promotes the project • Allocates his or her time to project • Provides resources	• Make a presentation about the objectives of the project and the proposed benefits to those executives who will benefit directly from the system • Create a prototype of the system to demonstrate its potential value
Organizational Management	Organizational managers: • Know about the project • Budget enough money for the project • Encourage users to accept and use the system	• Make a presentation to management about the objectives of the project and the proposed benefits • Market the benefits of the system using memos and organizational newsletters • Encourage the champion to talk about the project with his or her peers
System Users	Users: • Make decisions that influence the project • Perform hands-on activities for the project • Ultimately determine whether the project is successful by using or not using the system	• Assign users official roles on the project team • Assign users specific tasks to perform with clear deadlines • Ask for feedback from users regularly (e.g., at weekly meetings)

FIGURE 2-11 Some Important Stakeholders for Organizational Feasibility

PROJECT SELECTION

Once the feasibility analysis has been completed, it is submitted to the approval committee, along with a revised system request. The committee then decides whether to approve the project, decline the project, or table it until additional information is available. At the project level, the committee considers the value of the project by examining the business need (found in the system request) and the risks of building the system (presented in the feasibility analysis).



Size	What is the size? How many people are needed to work on the project?
Cost	How much will the project cost the organization?
Purpose	What is the purpose of the project? Is it meant to improve the technical infrastructure? Support a current business strategy? Improve operations? Demonstrate a new innovation?
Length	How long will the project take before completion? How much time will go by before value is delivered to the business?
Risk	How likely is it that the project will succeed or fail?
Scope	How much of the organization is affected by the system? A department? A division? The entire corporation?
Return on investment	How much money does the organization expect to receive in return for the amount the project costs?

FIGURE 2-12 Ways to Classify Projects

Chapter 4: Requirements Determination

INTRODUCTION

SDLC is the process by which an organization moves from the current system (often called the *as-is system*) to the new system (often called the *to-be system*). The output of planning, , is the system request, which provides general ideas for the to-be system, defines the project's scope, and provides the initial workplan. The analysis phase takes the general ideas in the system request and refines them into a detailed requirements definition, functional models, structural models, and behavioral models that together form the *system proposal.* The system proposal also includes revised project management deliverables, such as the feasibility analysis and the workplan.

REQUIREMENTS DETERMINATION

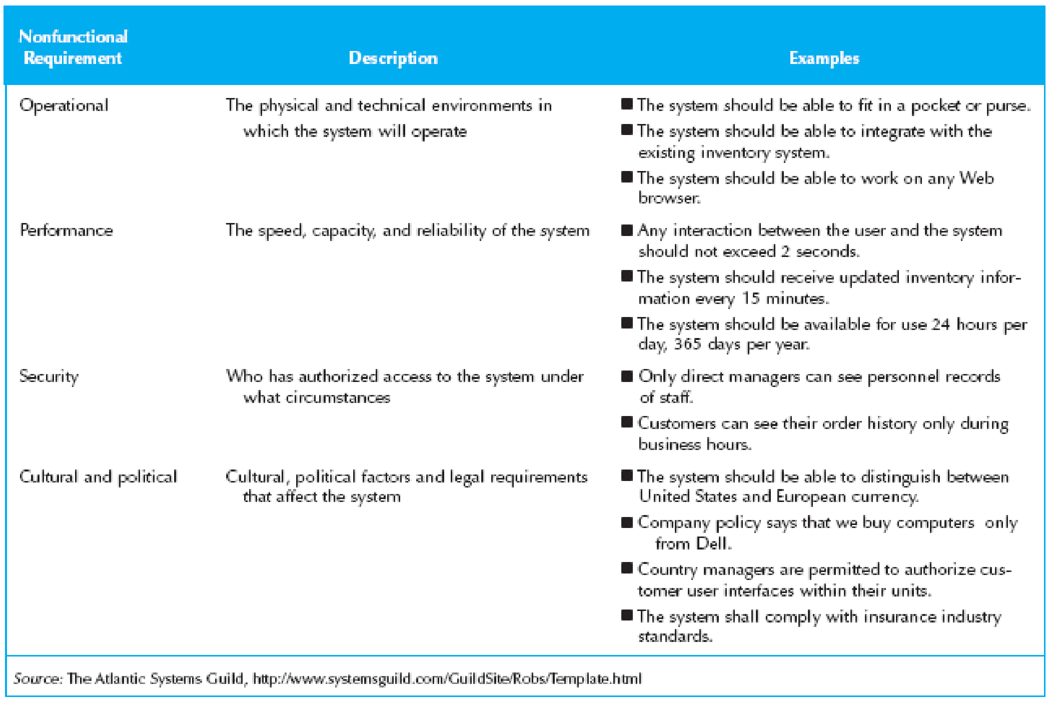
The purpose of the *requirements determination* step is to turn the very high-level explanation of the business requirements stated in the system request into a more precise list of requirements
that can be used as inputs to the rest of analysis (creating functional, structural, and behavioral models). This expansion of the requirements ultimately leads to the design phase.

Defining a Requirement

A *requirement* is simply a statement of what the system must do or what characteristic it must have. During analysis, requirements are written from the perspective of the businessperson,
and they focus on the "what" of the system. Because they focus on the needs of the business user, they are usually called *business requirements* (and sometimes user

Requirements can be either functional or nonfunctional in nature. A *functional requirement* relates directly to a process a system has to perform or information it needs to contain. For example, requirements that state that a system must have the ability to search for available inventory or to report actual and budgeted expenses are functional requirements.

Nonfunctional requirements refer to behavioral properties that the system must have, such as performance and usability. The ability to access the system using a Web browser is considered a nonfunctional requirement.Nonfunctional requirements may influence the rest of analysis (functional, structural, and behavioral models) but often do so only indirectly; nonfunctional requirements are used primarily in design when decisions are made about the user interface, the hardware and software, and the system's underlying physical architecture.

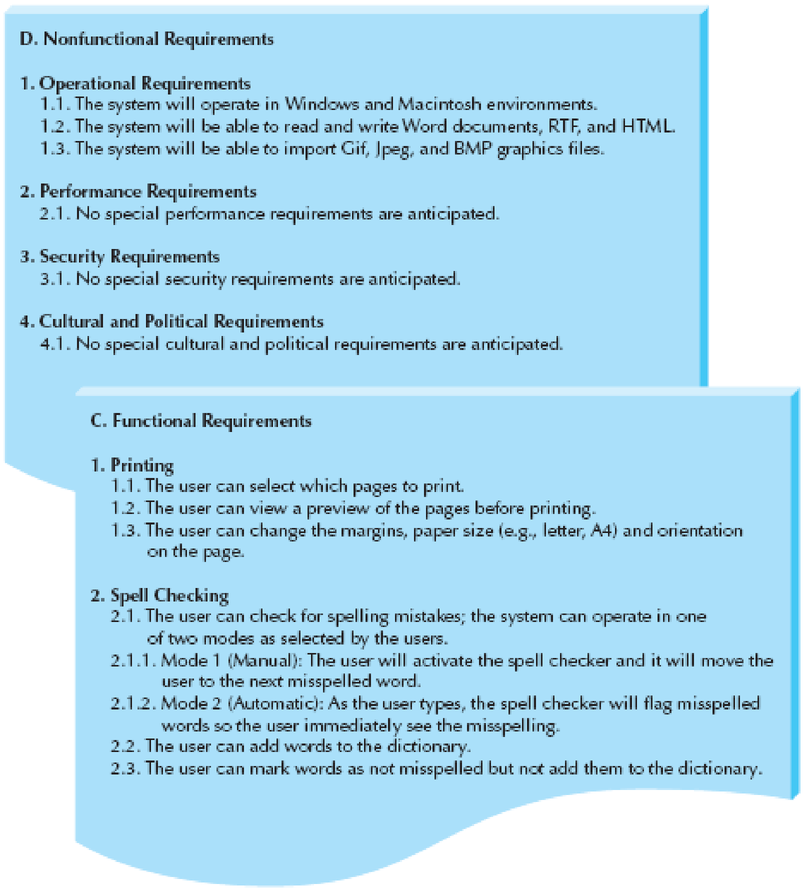


Nonfunctional Requirement	Description	Examples
Operational	The physical and technical environments in which the system will operate	■ The system should be able to fit in a pocket or purse. ■ The system should be able to integrate with the existing inventory system. ■ The system should be able to work on any Web browser.
Performance	The speed, capacity, and reliability of the system	■ Any interaction between the user and the system should not exceed 2 seconds. ■ The system should receive updated inventory information every 15 minutes. ■ The system should be available for use 24 hours per day, 365 days per year.
Security	Who has authorized access to the system under what circumstances	■ Only direct managers can see personnel records of staff. ■ Customers can see their order history only during business hours.
Cultural and political	Cultural, political factors and legal requirements that affect the system	■ The system should be able to distinguish between United States and European currency. ■ Company policy says that we buy computers only from Dell. ■ Country managers are permitted to authorize customer user interfaces within their units. ■ The system shall comply with insurance industry standards.

Source: The Atlantic Systems Guild, http://www.systemsguild.com/GuildSite/Robs/Template.html

Requirements Definition

The requirements definition report—usually just called the *requirements definition*—is a straightforward text report that simply lists the functional and nonfunctional requirements in an outline format. Figure 4-2 shows a sample requirements definition for a word processing program designed to compete against software such as Microsoft Word.


D. Nonfunctional Requirements

1. Operational Requirements
   1.1. The system will operate in Windows and Macintosh environments.
   1.2. The system will be able to read and write Word documents, RTF, and HTML.
   1.3. The system will be able to import Gif, Jpeg, and BMP graphics files.

2. Performance Requirements
   2.1. No special performance requirements are anticipated.

3. Security Requirements
   3.1. No special security requirements are anticipated.

4. Cultural and Political Requirements
   4.1. No special cultural and political requirements are anticipated.

C. Functional Requirements

1. Printing
   1.1. The user can select which pages to print.
   1.2. The user can view a preview of the pages before printing.
   1.3. The user can change the margins, paper size (e.g., letter, A4) and orientation on the page.

2. Spell Checking
   2.1. The user can check for spelling mistakes; the system can operate in one of two modes as selected by the users.
   2.1.1. Mode 1 (Manual): The user will activate the spell checker and it will move the user to the next misspelled word.
   2.1.2. Mode 2 (Automatic): As the user types, the spell checker will flag misspelled words so the user immediately see the misspelling.
   2.2. The user can add words to the dictionary.
   2.3. The user can mark words as not misspelled but not add them to the dictionary.

FIGURE 4-2
Sample Requirements Definition

Determining Requirements

Determining requirements for the requirements definition is both a business task and an information technology task.
the most effective approach is to have both business people and analysts working together to determine business requirements. Sometimes, however, users don't know exactly what they want, and analysts need to help them discover their needs. Three kinds of techniques have become popular to help analysts do this: *business process automation (BPA)*, *business process improvement (BPI)*, and *business process reengineering (BPR)*.Analysts can use these tools when they need to guide the users in explaining what is wanted from a system.
Although BPA, BPI, and BPR enable the analyst to help users create a vision for the new system, they are not sufficient for extracting information about the detailed business requirements
that are needed to build it. Therefore, analysts use a portfolio of requirements-gathering techniques to acquire information from users. The analyst has many gathering techniques from which to choose: *interviews*, *questionnaires*, *observation*, *JAD*, (*joint application development*)
and *document analysis*. The information gathered using these techniques is critically analyzed and used to craft the requirements definition report. The final section of this chapter describes each of the requirements-gathering techniques in greater depth.

Creating a Requirements Definition

Creating a requirements definition is an iterative and ongoing process whereby the analyst collects information with requirements-gathering techniques (e.g., interviews, document analysis), critically analyzes the information to identify appropriate business requirements for the system, and adds the requirements to the requirements definition report. The requirements definition is kept up to date so that the project team and business users can refer to it and get a clear understanding of the new system.

To create a requirements definition, the project team first determines the kinds of functional and nonfunctional requirements that they will collect about the system (of course, these may change over time). These become the main sections of the document. Next, the analysts use a variety of requirements-gathering techniques (e.g., interviews, observation) to collect information, and they list the business requirements that were identified from that information. Finally, the analysts work with the entire project team and the business users to verify, change, and complete the list and to help prioritize the importance of the requirements that were identified.

This process continues throughout analysis, and the requirements definition evolves over time as new requirements are identified and as the project moves into later phases of the SDLC.