#### *W1.Safari.References*

***Ctrl+Click the following links to go to their place in this document***

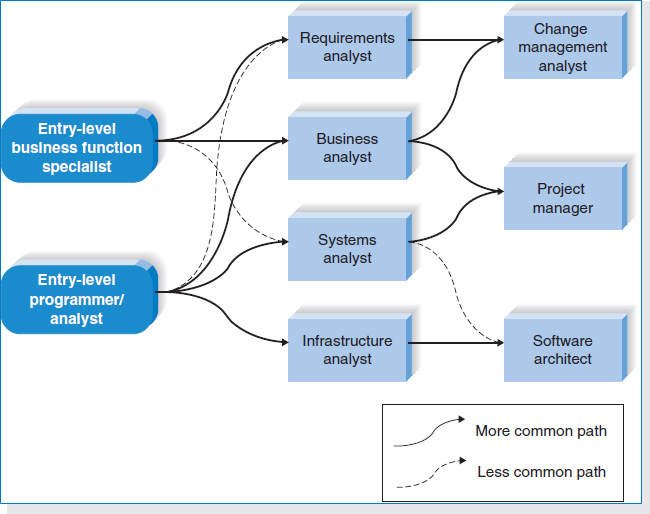
* [***Four Fundamental Phases of the SDLC***](#_Four_Fundamental_Phases)
* [***Object-Oriented Systems Analysis and Design (OOSAD)***](#_Object-Oriented_Systems_Analysis)
* [***Unified Modeling Language (UML)***](#_Unified_Modeling_Language)
* [***Unified Process (UP)***](#_Unified_Process_(UP))

## Four Fundamental Phases of the SDLC

<https://learning.oreilly.com/library/view/system-analysis-and/9781118057629/08_chap01.html>

## THE SYSTEMS DEVELOPMENT LIFE CYCLE

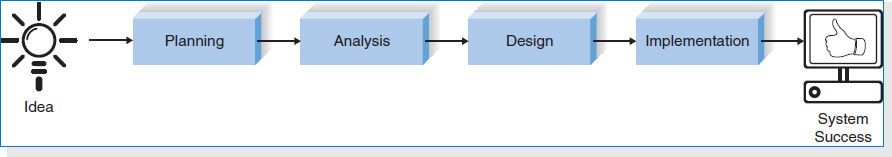
In many ways, building an information system is similar to building a house. First, the owner describes the vision for the house to the developer. Second, this idea is transformed into sketches and drawings that are shown to the owner and refined (often, through several drawings, each improving on the other) until the owner agrees that the pictures depict what he or she wants. Third, a set of detailed blue prints is developed that presents much more specific information about the house (e.g., the layout of rooms, placement of plumbing fixtures and electrical outlets, and so on). Finally, the house is built following the blueprints—and often with some changes and decisions made by the owner as the house is erected.



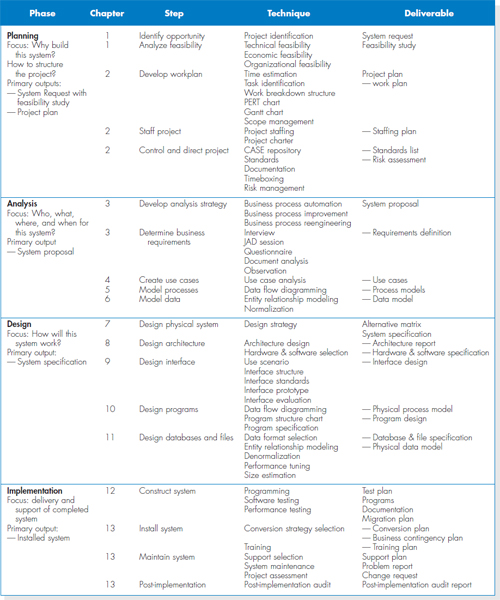
**FIGURE 1-1** Career Paths for System Developers

Building an information system using the SDLC follows a similar set of four fundamental *phases*: planning, analysis, design, and implementation ([Figure 1-2](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118057629/navPoint-20#fig1-2)). Each phase is itself composed of a series of *steps*, which rely on *techniques* that produce *deliverables* (specific documents and files that explain various elements of the system). [Figure 1-3](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118057629/navPoint-20#fig1-3) provides more detail on the steps, techniques, and deliverables that are included in each phase of the SDLC and outlines how these topics are covered in this textbook.

[Figures 1-2](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118057629/navPoint-20#fig1-2) and [1-3](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118057629/navPoint-20#fig1-3) suggest that the SDLC phases proceed in a logical path from start to finish. In some projects, this is true. In many projects, however, the project team moves through the steps consecutively, incrementally, iteratively, or in other patterns. Different projects may emphasize different parts of the SDLC or approach the SDLC phases in different ways, but all projects have elements of these four phases.



**FIGURE 1-2** The Systems Development Life Cycle



**FIGURE 1-3** Systems Development Life Cycle Phases

For now, there are two important points to understand about the SDLC. First, you should get a general sense of the phases and steps that IS projects move through and some of the techniques that produce certain deliverables. In this section, we provide an overview of the phases, steps, and some of the techniques that are used to accomplish the steps. Second, it is important to understand that the SDLC is a process of *gradual refinement*. The deliverables produced in the analysis phase provide a general idea what the new system will do. These deliverables are used as input to the design phase, which then refines them to produce a set of deliverables that describes in much more detailed terms exactly how the system should be built. These deliverables in turn are used in the implementation phase to guide the creation of the actual system. Each phase refines and elaborates on the work done previously.

### 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 generating the request (called the *project sponsor*) to conduct a feasibility analysis. The *feasibility analysis* examines key aspects of the proposed project:
   * The 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.

1. Once the project is approved, it enters *project management*. During project management, the *project manager* creates a *work plan*, 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* that describes how the project team will go about developing the system.

#### Analysis

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. (See [Figure 1-3](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118057629/navPoint-20#fig1-3).) During this phase, the project team investigates any current system(s), identifies improvement opportunities, and develops a concept for the new system. This phase has three steps:

1. An *analysis strategy* is developed to guide the project team's efforts. Such a strategy usually includes a study of the current system (called the *as-is system*) and its problems, and envisioning ways to design a new system (called the *to-be system*).
2. The next step is *requirements gathering* (e.g., through interviews, group workshops, or questionnaires). The analysis of this information—in conjunction with input from the 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* that describes how the business will operate if the new system were developed. The set 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 will decide whether the project should continue to move forward.

The system proposal is the initial deliverable that describes what business requirements the new system should meet. Because it is really the first step in the design of the new system, some experts argue that it is inappropriate to use the term *analysis* as the name for this phase; some argue a better name would be *analysis and initial design*. Because most organizations continue to use the name *analysis* for this phase, we will use it in this book as well. It is important to remember, however, that the deliverable from the analysis phase is both an analysis and a high-level initial design for the new system.

#### Design

The *design phase* decides *how* the system will operate in terms of the hardware, software, and network infrastructure that will be in place; the user interface, forms, and reports that will be used; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system are 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* must be determined. This clarifies whether the system will be developed by the company's own programmers, whether its development 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 that describes the hardware, software, and network infrastructure that will be used. In most cases, the system will add to or change the infrastructure that already exists in the organization. The *interface design* specifies how the users will move through the system (e.g., by navigation methods such as menus and on-screen buttons) and the forms and reports that the system will use.
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.

This collection of deliverables (architecture design, interface design, database and file specifications, and program design) is the *system specification* that is used by the programming team for implementation. At the end of the design phase, the feasibility analysis and project plan are reexamined and revised, and another decision is made by the project sponsor and approval committee about whether to terminate the project or continue. (See [Figure 1-3](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118057629/navPoint-20#fig1-3).)

### Implementation

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 and installed). 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 that it performs as designed. Since the cost of fixing bugs can be immense, testing is one of the most critical steps in implementation. Most organizations spend more time and attention on testing than on writing the programs in the first place.
2. The system is installed. *Installation* is the process by which the old system is turned off and the new one is turned on. There are several approaches that may be used to convert from the old to the new system. One of the most important aspects of conversion is the *training plan*, used to teach users how to use the new system and help manage the changes caused by the new system.
3. The analyst team establishes a *support plan* for the system. This plan usually includes a formal or informal post-implementation review, as well as a systematic way for identifying major and minor changes needed for the system.

## Object-Oriented Systems Analysis and Design (OOSAD)

<https://learning.oreilly.com/library/view/systems-analysis-and/9781118037423/05_chapter001.html#ch001-sec024>

Object-oriented approaches to developing information systems, technically speaking, can use any of the traditional methodologies. However, the object-oriented approaches are most associated with a phased development RAD or agile 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.

According to the creators of the Unified Modeling Language (UML), Grady Booch, Ivar Jacobson, and James Rumbaugh,[16](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118037423/navPoint-25#fn.0016) any modern object-oriented approach to developing information systems must be use-case driven, architecture-centric, and 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 business process at a time. In contrast, the process model diagrams used by traditional structured and RAD methodologies are far more complex because they require the systems analyst and user to develop models of the entire system. With traditional methodologies, each system is decomposed into a set of subsystems, which are, in turn, decomposed into further subsystems, 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, a use case focuses on only one business process at a time, so developing models is much simpler.[17](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118037423/navPoint-25#fn.0017)

### Architecture-centric

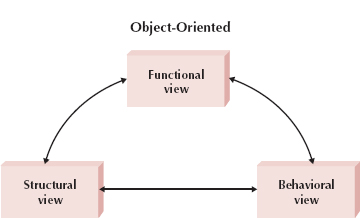
Any modern approach to systems analysis and design should be architecture-centric. *Architecture-centric* means that the underlying software architecture of the evolving system specification drives the specification, construction, and documentation of the system. Modern object-oriented systems analysis 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 iterates over each of and among the views. That is, as the analyst better understands the structural and behavioral views, the analyst uncovers 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 1-16](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9781118037423/navPoint-18#ch001-f016)). As each increment and iteration is completed, a more-complete representation of the user's real functional requirements is uncovered.

### Benefits of Object-Oriented Systems Analysis and Design

Concepts in the object-oriented approach enable analysts to break a complex system into smaller, more-manageable modules, work on the modules individually, and easily piece the modules back together to form an information system. This modularity makes systems development easier to grasp, easier to share among members of a project team, and easier to communicate to users, who are needed to provide requirements and confirm how well the system meets the requirements throughout the systems development process. By modularizing systems development, the project team actually is creating reusable pieces that can be plugged into other systems efforts or used as starting points for other projects. Ultimately, this can save time because new projects don't have to start completely from scratch.



**FIGURE 1-16** Iterative and Incremental Development

Many people argue that “object-think” is a much more realistic way to think about the real world. Users typically do not think in terms of data or process; instead, they see their business as a collection of logical units that contain both, so communicating in terms of objects improves the interaction between a user and an analyst or developer.

## [Unified Modeling Language (UML)](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/book/software-engineering-and-development/object/9780124186736/chapter-1dot-introduction/chp001_html" \l "X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODAxMjQxODY3MzYlMkZzdDAwMzBfaHRtbCZxdWVyeT1jaG9vc2UlMjBoYXJ)

<https://learning.oreilly.com/library/view/Systems+Analysis+and+Design+with+UML,+4th+Edition/9781118037423/05_chapter001.html#ch001-sec033>

Some developers believe that UML is a methodology, maybe because of the “M” in the acronym. However, that is not true: UML means Unified Modeling Language, and it is therefore a language that can be used to describe things.

Knowing a language does not necessarily imply the ability to produce useful artifacts. For example, English is a language, but someone who knows how to speak English does not necessarily know how to write good poetry or how to make good speeches. Besides the language syntax, there are knowledge and techniques of best practices that greatly help poets and speakers to place the elements of the language in an order and structure that is adequate to produce the expected results.

The UML language has been under development since James Rumbaugh and Grady Booch joined forces at Rational Software and started to unify their already well-known diagrammatic notations and processes. Later, Ivar Jacobson joined the group and added his use cases and other notations to the unified language that was under development.

UML is constantly being revised and currently has the following three families of diagrams:

• Structure diagrams: Includes package, class, objects, composite structure, component, profile, and deployment diagrams. They are used to define what must be implemented in the system in terms of components. They are useful to specify the part of the system architecture that is time independent.

• Behavior diagrams: Includes use case, activity, and state machine diagrams. They emphasize what must happen in the system or business process. They are used to describe the functionality of the system.

• Interaction diagrams: Includes communication, sequence, timing, and interaction overview diagrams. These are a subset of behavior diagrams and describe the control flow between different components of the system.

Not every diagram must be used during the development of a system. Only those that represent useful information for the project are recommended. This book emphasizes the use of the activity, machine state, use case, sequence, communication, and class diagrams for modeling information systems. However, other diagrams can be useful depending on the features of the system being modeled. For more information on the different UML diagrams, the book by [Miles and Hamilton (2006)](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/bib001_html#FUR56) may be consulted.

## Unified Process (UP)

<https://learning.oreilly.com/library/view/Systems+Analysis+and+Design+with+UML,+4th+Edition/9781118037423/05_chapter001.html#ch001-sec029>

The techniques presented in this book are compatible with the Unified Process, which is heavily based (although not necessarily) on UML.

UP was also proposed by the three amigos, Grady Booch, James Rumbaugh, and Ivar Jacobson ([Jacobson, Booch, and Rumbaugh, 1999](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/bib001_html#FUR40)), as the result of their extensive experience.

This process is based in the following principles:

• Use case driven: The development is planned and organized over a list of use cases.

• Architecture centered: The development process leads to the construction of a system architecture that allows the implementation of the requirements. That architecture is based on the identification of a structure that is iteratively built from a conceptual model.

• Iterative and incremental: Development is divided into iterations or development cycles. At each iteration, new features are added to the system architecture, or corrected/refined, leaving it more complete and closer to the final desired system.

• Risk oriented: The elements of greater risk for a project are addressed early. For instance, critical use cases are identified, detailed, and implemented before the others.

UP includes in its disciplines the main activities related to software development. Those activities have different levels or emphasis during the four major UP phases: Inception, Elaboration, Construction, and Transition ([Figure 1.1](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0035_html#F0010)). Although sequential in time, those phases must not be confused with the waterfall phases ([Royce, 1970](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/bib001_html#FUR71)). In the waterfall model, requirements specification must be completed before design, design must be completed before construction, and so on. In UP, requirements specification, design, construction, and other activities are performed in all phases with different emphasis.

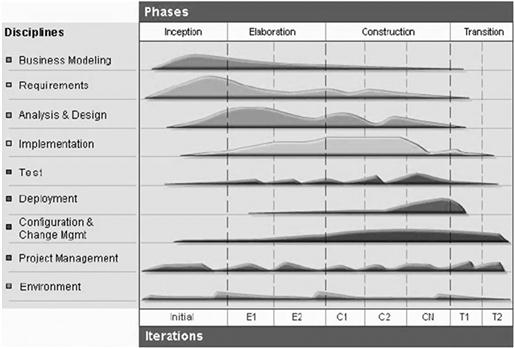


Figure 1.1 The emphasis of the different disciplines during the four phases of the Rational Unified Process (RUP®).[2](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0045_html#FN5), [3](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0045_html#FN6)

Inception is the first phase of UP, in which the main requirements are discovered and the extension of the system is understood. The output of this phase usually consists of a preliminary conceptual model ([Section 3.6](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0070_chp003_html#S0105)); a requirements document, usually in the form of a list of high-level use cases ([Section 3.3](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0025_chp003_html#S0020_CHP003)) and supplementary specifications ([Section 3.5.8](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0070_chp003_html#S0100)); and a development schedule based on the use case list ([Section 4.3](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0035_chp004_html#S0080_CHP004)). Additionally, a list of high-importance risks and their mitigation plans may be created, as well as other plans attending the special needs of the project. But those aspects are out of the scope of this book, which will concentrate on modeling techniques, not management. Only iteration planning based on high-level use cases is presented in detail.

The Elaboration phase includes a more detailed requirements analysis, which is performed by expanding the use cases, that is, writing the sequence of steps that characterizes each of their possible flows. The conceptual model is refined after each use case is expanded. Depending on the priority of the use cases, it is expected that the number of changes applied to the software architecture decreases as the project proceeds during Elaboration.

During the Construction phase most of the code production and test activities are performed. It is expected that the Elaboration phase produces an architecture sufficiently stable so that its refactoring will be minimized during this phase.

The Transition phase consists typically of the final tests and the delivery of the system to its users including possibly its installation and data migration. During this phase, the system will be deployed, possibly replacing an existing system (manual or automatic).

The Elaboration and Construction phases are performed in iterations. An iteration may have as an objective developing one or more use cases, implementing change requests, or mitigating selected risks. During an iteration, use cases are expanded and the information learned from them is incrementally incorporated in the product. It is expected that the Elaboration phase deals with the major risks of the system, as well as with the more complex or risky use cases that affect the system architecture significantly. On the other hand, the Construction phase concentrates on producing code for the whole application and implementing change requests.

UP is usually understood as a prescriptive process. But it may also be performed as an agile method, with few artifacts. Two popular agile implementations of UP are AUP[4](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0045_html#FN2) ([Ambler & Jeffries, 2002](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/bib001_html#FUR7)) and OpenUP[5](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0045_html#FN3) ([Kroll & MacIsaac, 2006](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/bib001_html#FUR48)). An agile process is one that prioritizes:[6](http://proquestcombo.safaribooksonline.com.ezproxy.umuc.edu/9780124186736/st0045_html#FN4)

• People and iterations over tools and processes.

• Working software over comprehensive documentation.

• Customer collaboration over contract negotiation.

• Responding to change over following a plan.

In order to obtain such agility, all documentation must be directed to software production. Each activity performed by the developer must have a very clear goal and a precise use, aiming always toward the production of code that meets the requirements in the best way possible and in the shortest reasonable time. Software is designed with two goals in mind: understanding client needs, and producing a viable solution to those needs. In order to help people adequately communicate their needs and solutions, different artifacts, such as diagrams, may be created; diagrams are more useful when they allow code to be automatically generated from them.