DENNIS • WIXOM • TEGARDEN

SYSTEMS ANALYSIS & DESIGN

An Object-Oriented Approach with UML

5TH EDITION

WILEY

Visible®
SYSTEMS CORPORATION

Visible Analyst Student Edition

Educating tomorrow's developers today

Visible Analyst is a "hands-on" tool for teaching students all aspects of analysis and design including dynamic rules, consistency checking, managing change, and understanding the integration issues across an IT project. Visible Analyst prepares students to enter the IT world as business or data architects, analysts, designers, and modelers.

Visit us at **www.visible.com** to learn more.

YOU CAN Start Today with the Visible Analyst!

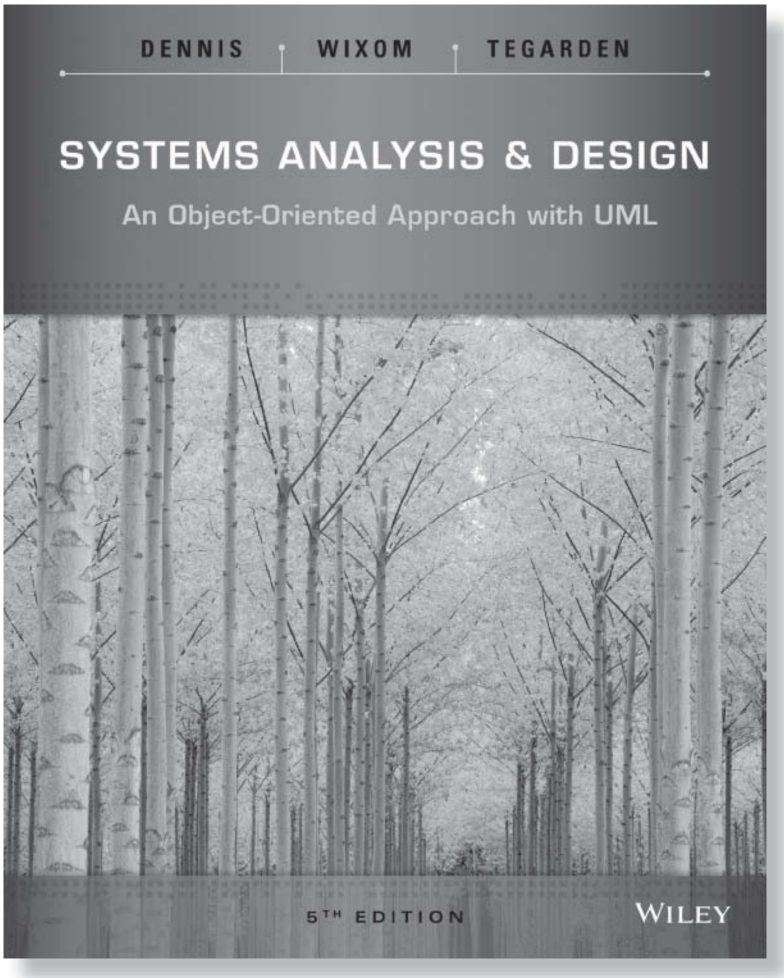
Only takes 2 minutes to install!
Save... 33% discount!

Please visit **http://store.visible.com/Wiley.aspx** to purchase and register with your information (see below) and obtain a valid license for your student edition of the software. To purchase the discounted software you will need to enter the following code:

978112014

Email support is provided to all registered students at support@visible.com. Your registration includes

- the latest release of the Visible Analyst Student Edition (software)
- the Visible Analyst eTutorial
- a preloaded Sample Instructional Project
- access to Webcast "How to" and "Get Started" Instructional Videos.


DENNIS WIXOM TEGARDEN
SYSTEMS ANALYSIS & DESIGN
An Object-Oriented Approach with UML
5TH EDITION
WILEY

Disclaimer: The publisher of the textbook does not sponsor, review, or make decisions about Visible Analyst software, and will not be responsible for, or involved in, any changes to the software.

System Analysis & Design

An Object-Oriented Approach with UML

Fifth Edition

Alan Dennis
Indiana University

Barbara Haley Wixom
Massachusetts Institute of Technology

David Tegarden
Virginia Tech
With contributions by Elaine Seeman,
East Carolina University

WILEY

VP & EXECUTIVE PUBLISHER:	Don Fowley
EXECUTIVE EDITOR:	Beth Lang Golub
CONTENT EDITOR:	Mary O'Sullivan
ASSOCIATE EDITOR:	Ellen Keohane
MARKETING MANAGER:	Christopher Ruel
ASSOCIATE PRODUCTION MANAGER:	Joyce Poh
DESIGNER:	Wendy Lai

Cover Image: © Christopher Boswell/Shutterstock

This book was set in 10/12 Minion pro by Aptara and printed and bound by Courier Kendallville. The cover was printed by Courier Kendallville.

This book is printed on acid-free paper. ∞

Founded in 1807, John Wiley & Sons, Inc. has been a valued source of knowledge and understanding for more than 200 years, helping people around the world meet their needs and fulfill their aspirations. Our company is built on a foundation of principles that include responsibility to the communities we serve and where we live and work. In 2008, we launched a Corporate Citizenship Initiative, a global effort to address the environmental, social, economic, and ethical challenges we face in our business. Among the issues we are addressing are carbon impact, paper specifications and procurement, ethical conduct within our business and among our vendors, and community and charitable support. For more information, please visit our website: www.wiley.com/go/citizenship.

Copyright © 2015, 2012, 2009 John Wiley & Sons, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923 (Web site: www.copyright.com). Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201) 748-6011, fax (201) 748-6008, or online at: www.wiley.com/go/permissions.

Evaluation copies are provided to qualified academics and professionals for review purposes only, for use in their courses during the next academic year. These copies are licensed and may not be sold or transferred to a third party. Upon completion of the review period, please return the evaluation copy to Wiley. Return instructions and a free of charge return shipping label are available at: www.wiley.com/go/returnlabel. If you have chosen to adopt this textbook for use in your course, please accept this book as your complimentary desk copy. Outside of the United States, please contact your local sales representative.

Library of Congress Cataloging-in-Publication Data

Dennis, Alan.

Systems analysis & design: an object-oriented approach with UML/Alan Dennis, Indiana University, Barbara Haley Wixom, Massachusetts Institute of Technology, David Tegarden, Virginia Tech; with contributions by Elaine Seeman, East Carolina University.–Fifth edition.

pages cm

Includes bibliographical references and index.

ISBN 978-1-118-80467-4 (pbk. : alk. paper)

1. System analysis. 2. System design. 3. UML (Computer science) I. Wixom, Barbara Haley, 1969-II. Tegarden, David Paul. III. Seeman, Elaine. IV. Title. V. Title: System analysis and design.

QA402.D395 2015

004.2'1–dc23

2014048338

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PREFACE

PURPOSE OF THIS BOOK

Systems Analysis and Design (SAD) is an exciting, active field in which analysts continually learn new techniques and approaches to develop systems more effectively and efficiently. However, there is a core set of skills that all analysts need to know—no matter what approach or methodology is used. All information systems projects move through the four phases of planning, analysis, design, and implementation; all projects require analysts to gather requirements, model the business needs, and create blueprints for how the system should be built; and all projects require an understanding of organizational behavior concepts like change management and team building. Today, the cost of developing modern software is composed primarily of the cost associated with the developers themselves and not the computers. As such, object-oriented approaches to developing information systems hold much promise in controlling these costs.

Today, the most exciting change to systems analysis and design is the move to object-oriented techniques, which view a system as a collection of self-contained objects that have both data and processes. This change has been accelerated through the creation of the Unified Modeling Language (UML). UML provides a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation.

This book captures the dynamic aspects of the field by keeping students focused on doing SAD while presenting the core set of skills that we feel every systems analyst needs to know today and in the future. This book builds on our professional experience as systems analysts and on our experience in teaching SAD in the classroom.

This book will be of particular interest to instructors who have students do a major project as part of their course. Each chapter describes one part of the process, provides clear explanations on how to do it, gives a detailed example, and then has exercises for the students to practice. In this way, students can leave the course with experience that will form a rich foundation for further work as a systems analyst.

OUTSTANDING FEATURES

A Focus on Doing SAD

The goal of this book is to enable students to do SAD—not just read about it, but understand the issues so that they can actually analyze and design systems. The book introduces each major technique, explains what it is, explains how to do it, presents an example, and provides **Your Turn** opportunities with each chapter for students to practice each new technique before they do it for real in a project. The **Your Turn** boxes are posted online at www.wiley.com/college/dennis. After reading each chapter, the student will be able to perform that step in the system development process.

Rich Examples of Success and Failure

This book has a running online case study (accessible from www.wiley.com/go/dennis/casestudy) about a fictitious health care company called Patterson Superstore. Each chapter of the case study shows how the concepts are applied in situations at Patterson Superstore. In this way, the running case serves as a template that students can apply to their own work. Each chapter also includes numerous **Concepts in Action** boxes, which are posted online at www.wiley.com/college/dennis. These boxes describe how real companies succeeded—and failed—in performing the activities in the chapter. Many of these examples are drawn from our own experiences as systems analysts.

Real World Focus

The skills that students learn in a systems analysis and design course should mirror the work that they ultimately will do in real organizations. We have tried to make this book as "real" as possible by building extensively on our experience as professional systems analysts for organizations, such as Arthur Andersen, IBM, the U.S. Department of Defense, and the Australian Army. We have also worked with a diverse industry advisory board of IS professionals and consultants in developing the book and have incorporated their stories, feedback, and advice throughout. Many students who use this book will eventually use the skills on the job in a business environment, and we believe they will have a competitive edge in understanding what successful practitioners feel is relevant in the real world.

Project Approach

We have presented the topics in this book in the order in which an analyst encounters them in a typical project. Although the presentation is necessarily linear (because students have to learn concepts in the way in which they build on each other), we emphasize the iterative, complex nature of SAD as the book unfolds. The presentation of the material should align well with courses that encourage students to work on projects because it presents topics as students need to apply them.

WHAT'S NEW IN THIS EDITION

- A completely new, expanded case study on an integrated health clinic delivery system has been written to accompany the fifth edition. The entire case study is posted online. At the end of each chapter in the text, a short synopsis of the case is provided.
- The text has been streamlined to focus on the essentials and therefore, to enhance student understanding. Selected materials like the "Your Turn" and "Concepts in Action" boxes have been moved online and can be accessed at www.wiley.com/college/dennis.
- Throughout the book, there is a greater emphasis on verifying, validating, and testing, as well as the incremental and iterative development of systems.
- In Chapter 2, there is more content on Agile techniques, including scrum meetings, product backlog, and sprints.
- In Chapter 3, we have increased focus on software quality and user stories.
- We have added new examples throughout the book and clarified explanations to help students learn some of the more difficult concepts.

- Chapter 10 includes more coverage of mobile computing, including specifics on navigation, input, and output. This chapter also has a new section on games, multidimensional information visualization, augmented reality, and virtual reality.
- Chapter 11 includes new material on ubiquitous computing and the Internet of Things.
- Testing has been expanded in Chapter 12.

ORGANIZATION OF THIS BOOK

This book is loosely organized around the phases and workflows of the enhanced Unified Process. Each chapter has been written to teach students specific tasks that analysts need to accomplish over the course of a project, and the deliverables that will be produced from the tasks. As students complete the chapters, they will realize the iterative and incremental nature of the tasks in object-oriented systems development.

Chapter 1 introduces the SDLC, systems development methodologies, roles and skills needed for a systems analyst, the basic characteristics of object-oriented systems, object-oriented systems analysis, the Unified Process, and the UML. Chapter 2 presents topics related to the project management workflow of the Unified Process, including project identification, system request, feasibility analysis, project selection, traditional project management tools (including work breakdown structures, network diagrams, and PERT analysis), project effort estimation using use-case points, evolutionary work breakdown structures, iterative workplans, scope management, timeboxing, risk management, and staffing the project. Chapter 2 also addresses issues related to the Environment and Infrastructure management workflows of the Unified Process.

Part One focuses on creating analysis models. Chapter 3 introduces students to an assortment of requirements analysis strategies a variety of requirements-gathering techniques that are used to determine the functional and nonfunctional requirements of the system, and to a system proposal. Chapter 4 focuses on constructing business process and functional models using use-case diagrams, activity diagrams, and use-case descriptions. Chapter 5 addresses producing structural models using CRC cards, class diagrams, and object diagrams. Chapter 6 tackles creating behavioral models using sequence diagrams, communication diagrams, behavioral state machines, and CRUDE analysis and matrices. Chapters 4 through 6 also cover the verification and validation of the models described in each chapter.

Part Two addresses design modeling. In Chapter 7, students learn how to verify and validate the analysis models created during analysis modeling and to evolve the analysis models into design models via the use of factoring, partitions, and layers. The students also learn to create an alternative matrix that can be used to compare custom, packaged, and outsourcing alternatives. Chapter 8 concentrates on designing the individual classes and their respective methods through the use of contracts and method specifications. Chapter 9 presents the issues involved in designing persistence for objects. These issues include the different storage formats that can be used for object persistence, how to map an object-oriented design into the chosen storage format, and how to design a set of data access and manipulation classes that act as a translator between the classes in the application and the object persistence. This chapter also focuses on the nonfunctional requirements that impact the data management layer. Chapter 10 presents the design of the human–computer interaction layer, where students learn how to design user interfaces using use scenarios, windows navigation diagrams, storyboards, windows layout diagrams, user interface prototypes, real use cases, interface standards, and user interface templates; to perform user interface evaluations using heuristic evaluation, walkthrough evaluation, interactive evaluation, and formal usability testing; and to address nonfunctional requirements such as user interface layout, content awareness, aesthetics, user experience, and consistency. This chapter also addresses issues related to mobile computing, social media, games, multidimensional information visualizations, immersive environments, and international and cultural issues with regard to user interface design. Chapter 11 focuses on the physical architecture and infrastructure design, which includes deployment diagrams and hardware/software specification. In today's world, this also includes issues related to cloud computing, ubiquitous computing, the Internet of things, and green IT. This chapter, like the previous design chapters, covers the impact that nonfunctional requirements can have on the physical architecture layer.

Part Three provides material that is related to the construction, installation, and operations of the system. Chapter 12 focuses on system construction, where students learn how to build, test, and document the system. Installation and operations are covered in Chapter 13, where students learn about the conversion plan, change management plan, support plan, and project assessment. Additionally, these chapters address the issues related to developing systems in a flat world, where developers and users are distributed throughout the world.

SUPPLEMENTS www.wiley.com/college/dennis

Instructor Book Companion Website

- **PowerPoint slides:** Instructors can tailor the slides to their classroom needs. Students can use them to guide their reading and studying activities.
- **Test Bank:** Includes a variety of questions ranging from multiple-choice, true/false, and short answer questions. A computerized, **Respondus** version of the Test Bank is also available.
- **Instructor's Manual:** Provides resources to support the instructor both inside and out of the classroom. The manual includes short experiential exercises that instructors can use to help students experience and understand key topics in each chapter. Short stories have been provided by people working in both corporate and consulting environments for instructors to insert into lectures to make concepts more colorful and real. Additional minicases for every chapter allow students to perform some of the key concepts that were learned in the chapter. Solutions to end of chapter questions and exercises are provided.

Student Book Companion Website

- A collection of **templates** and worksheets consisting of electronic versions of selected figures from the book.
- A completely new, expanded **case study** on an integrated health clinic delivery system has been written to accompany the fifth edition. This case study is online only. It can be accessed at www.wiley.com/go/dennis/casestudy.
- "Your Turn" and "Concepts in Action" boxes from the fourth edition have been moved online and can be accessed from the student companion site.

Wiley E-Text: Powered by VitalSource

This Wiley e-text offers students continuing access to materials for their course. Your students can access content on a mobile device, online from any Internet-connected computer, or by a computer via download. With dynamic features built into this e-text, students can search across content, highlight, and take notes that they can share with teachers and classmates.

Visible Analyst

Wiley has partnered with Visible Analyst to give students a discounted price for Visible Analyst software, an intuitive modeling tool for all aspects of traditional or object-oriented systems analysis and design. All new copies of the text will have a Key Code (printed on a page near the front of this text) that will provide a discount on Visible Analyst software. To obtain the software, students should visit http://store.visible.com/Wiley.aspx and enter their Key Code. Students who buy a new print text or digital e-book will receive one-third off the price of a downloadable edition of the software with a 6-month license. With the software, they will also receive tutorials, how-to videos, and a sample project. Students who buy used copies of this text may buy Visible Analyst at full price using the URL provided.

Project Management Software

You can download a 60-day trial of Microsoft Project Professional 2013 from the following Website: www.microsoft.com/en-us/evalcenter/evaluate-project-professional-2013. Note that Microsoft has changed its policy and no longer offers the 120-day trial previously available.

Another option now available to education institutions adopting this Wiley title is a free introductory 3-year membership for DreamSpark Premium. DreamSpark Premium is designed to provide the easiest and most inexpensive way for academic departments to make the latest Microsoft software available in labs, classrooms, and on student and instructor PCs. Microsoft Project software is available through this Wiley and Microsoft publishing partnership, free of charge with the adoption of any qualified Wiley title. Each copy of Microsoft Project is the full version of the software, with no time limitation, and can be used indefinitely for educational purposes. Contact your Wiley sales representative for details. For more information about the DreamSpark Premium program, contact drmspkna@Microsoft.com.

ACKNOWLEDGMENTS

Thanks to Elaine Seeman for her feedback on every chapter in this book as well as for her work writing the new online case study. We would like to thank the following reviewers for their helpful and insightful comments on the fifth edition: Mohammad Dadashzadeh, Oakland University; Xiaodong Deng, Oakland University; Thomas W. Dillon, James Madison University; Bryan Goda, University of Washington, Tacoma; Kathleen S. Hartzel, Duquesne University; Rajkumar Kempaiah, Stevens Institute of Technology; Sung-kwan Kim, University of Arkansas at Little Rock; Richard McCarthy, Quinnipiac University; Donald McCracken, Grantham University; Osama A. Morad, Southern New Hampshire University; Fred Niederman, Saint Louis University; Linda Plotnick, Jacksonville State University; Vladimir V. Riabov, Rivier University; Richard Schilhavy, Guilford College; Tod Sedbrook, University of Northern Colorado; Steven C. Shaffer, Penn State University; Michael Smith, Georgia Institute of Technology; and John Wetsch, Southern New Hampshire University.

We would also like to thank the following reviewers for their helpful and insightful comments on the first, second, third, and fourth editions: Evans Adams, Fort Lewis College; Murugan Anandarajon, Drexel University; Ron Anson, Boise State University; Noushin Ashrafi, University of Massachusetts, Boston; Dirk Baldwin, University of Wisconsin-Parkside; Robert Barker, University of Louisville; Qing Cao, University of Missouri–Kansas City; David Champion, DeVry University, Columbus, OH campus; Jeff Cummings, Indiana University; Junhua Ding, East Carolina University; Robert Dollinger, University of Wisconsin-Stevens Point; Abhijit Dutt, Carnegie Mellon University; Terry Fox, Baylor University; Ahmad Ghafarian, North Georgia College & State University; Donald Golden, Cleveland State University; Cleotilde Gonzalez, Carnegie Melon University; Daniel V. Goulet, University of Wisconsin–Stevens Point; Harvey Hayashi, Loyalist College of Applied Arts and Technology; Yujong Hwang, DePaul University; Scott James, Saginaw Valley State University; Zongliang Jiang, North Carolina A&T State University; Raymond Kirsch, La Salle University; Rajiv Kishore, State University of New York–Buffalo; Ravindra Krovi, University of Akron; Jean-Piere Kuilboer, University of Massachusetts, Boston; Gilliean Lee, Lander University; LeoLegorreta, California State University Sacramento; Diane Lending, James Madison University; Steve Machon, DeVry University; Fernando Maymí, West Point University; Daniel Mittleman, DePaulUniversity; Makoto Nakayama, DePaul University; Fred Niederman, Saint Louis University; Parasuraman Nurani, DeVry University; H. Robert Pajkowski, DeVry Institute of Technology, Scarborough, Ontario; June S. Park, University of Iowa; Graham Peace, West Virginia University; Tom Pettay, DeVry Institute of Technology, Columbus,Ohio; Selwyn Piramuthu, University of Florida; J. Drew Procaccino, Rider University; Neil Ramiller, Portland State University; Eliot Rich, University at Albany, State University of New York; Marcus Rothenberger, University of Wisconsin–Milwaukee; Carl Scott, University of Houston; Keng Siau,University of Nebraska–Lincoln; Iftikhar Sikder, Cleveland State University; Jonathan Trower, Baylor University; June Verner, Drexel University; Anna Wachholz, Sheridan College; Bill Watson, Indiana University-Purdue University Indianapolis; Randy S.Weinberg, Carnegie Mellon University; Eli J.Weissman, DeVry Institute of Technology, Long Island City, NY; Heinz Roland Weistroffer, Virginia Commonwealth University; Amy Wilson, DeVry Institute of Technology, Decatur, GA; Amy Woszczynski, Kennesaw State University; Vincent C. Yen, Wright State University; Fan Zhao, Florida Gulf Coast University; and Dan Zhu, Iowa State University.

CONTENTS

CHAPTER 1

Introduction to Systems Analysis and Design

Chapter 1 introduces the systems development life cycle (SDLC), the fundamental four-phase model (planning, analysis, design, and implementation) common to all information systems development projects. It describes the evolution of system development methodologies and discusses the roles and skills required of a systems analyst. The chapter then overviews the basic characteristics of object-oriented systems and the fundamentals of object-oriented systems analysis and design and closes with a description of the Unified Process and its extensions and the Unified Modeling Language.

OBJECTIVES

- Understand the fundamental systems development life cycle and its four phases
- Understand the evolution of systems development methodologies
- Be familiar with the different roles played by and the skills of a systems analyst
- Be familiar with the basic characteristics of object-oriented systems
- Be familiar with the fundamental principles of object-oriented systems analysis and design
- Be familiar with the Unified Process, its extensions, and the Unified Modeling Language

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. If you have taken a programming class or have programmed on your own, this probably sounds pretty simple. Unfortunately, it is not. A 1996 survey by the Standish Group found that 42 percent of all corporate IS projects were abandoned before completion. A similar study conducted in 1996 by the General Accounting Office found 53 percent of all U.S. government IS projects were abandoned. Unfortunately, many of the systems that are not abandoned are delivered to the users significantly late, cost far more than planned, and have fewer features than originally planned. For example, IAG Consulting reports that 80 percent of the projects were over time, 72 percent were over budget, and 55 percent contained less than the full functionality; Panorama Consulting Solutions reports that 54 percent of the ERP projects were over time, 56 percent were over budget, and 48 percent delivered less than 50 percent of the initial benefits; and an IBM study reports that 59 percent of the projects missed one or more of on time, within budget, and quality constraints.[1] Although we would like to promote this book as a silver bullet that will keep you from IS failures, we readily admit that a silver bullet that guarantees IS development success simply does not exist. Instead, this book provides you with several fundamental concepts and many practical techniques that you can use to improve the probability of success.

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. Being a systems analyst is one of the most interesting, exciting, and challenging jobs around. Systems analysts work with a variety of people and learn how they conduct business. Specifically, they work with a team of systems analysts, programmers, and others on a common mission. Systems analysts feel the satisfaction of seeing systems that they designed and developed make a significant business impact, knowing that they contributed unique skills to make that happen.

However, the primary objective of a systems analyst is not to create a wonderful system; instead, it is to create value for the organization, which for most companies means increasing profits (government agencies and not-for-profit organizations measure value differently). Many failed systems have been abandoned because the analysts tried to build a wonderful system without clearly understanding how the system would fit with an organization's goals, current business processes, and other information systems to provide value. An investment in an information system is like any other investment. The goal is not to acquire the tool, because the tool is simply a means to an end; the goal is to enable the organization to perform work better so that it can earn greater profits or serve its constituents more effectively.

This book introduces the fundamental skills a systems analyst needs. This pragmatic book discusses best practices in systems development; it does not present a general survey of systems development that covers everything about the topic. By definition, systems analysts do things and challenge the current way that organizations work. To get the most out of this book, you will need to actively apply to your own systems development project the ideas and concepts in the examples. This book guides you through all the steps for delivering a successful information system. By the time you finish the book, you won't be an expert analyst, but you will be ready to start building systems for real.

THE SYSTEMS DEVELOPMENT LIFE CYCLE

In many ways, building an information system is similar to building a house. First, the house (or the information system) starts with a basic idea. Second, this idea is transformed into a simple drawing that is shown to the customer and refined (often through several drawings, each improving on the last) until the customer agrees that the picture depicts what he or she wants. Third, a set of blueprints is designed that presents much more detailed information about the house (e.g., the type of water faucets or where the telephone jacks will be placed). Finally, the house is built following the blueprints, often with some changes directed by the customer as the house is erected.

The SDLC has a similar set of four fundamental *phases*: planning, analysis, design, and implementation. Different projects might emphasize different parts of the SDLC or approach the SDLC phases in different ways, but all projects have elements of these four phases. 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).

[1] For more information on the problem, see Capers Jones, *Patterns of Software System Failure and Success* (London: International Thompson Computer Press, 1996); KeithEllis, *Business Analysis Benchmark: The Impact of Business Requirements on the Success of Technology Projects* (2008). Retrieved May 2014 from IAG Consulting, www.iag.biz; H. H. Jorgensen, L. Owen, and A. Neus, *Making Change Work* (2008). Retrieved May 2014 from IBM, www.ibm.com; Panorama Consulting Solutions, *2012 ERP Report* (2012). Retrieved May 2014 from Panorama-Consulting.com.

For example, in applying for admission to a university, all students go through the same phases: information gathering, applying, and accepting. Each of these phases has steps; for example, information gathering includes steps such as searching for schools, requesting information, and reading brochures. Students then use techniques (e.g., Internet searching) that can be applied to steps (e.g., requesting information) to create *deliverables* (e.g., evaluations of different aspects of universities).

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. In this section, we describe the phases, the actions, and some of the techniques that are used to accomplish the steps at a very high level.

For now, there are two important points to understand about the SDLC. First, you should get a general sense of the phases and steps through which IS projects move and some of the techniques that produce certain deliverables. 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 of the shape of the new system. 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 will be built. These deliverables, in turn, are used in the implementation phase to produce 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 (e.g., from the marketing department, accounting department) 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 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.

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. During this phase, the project team investigates any current system(s), identifies opportunities for improvement, 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 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 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,* which describe how the business will operate if the new system is developed.
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.

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." Most organizations continue to use the name *analysis* for this phase, however, so we use it in this book as well. Just keep in mind 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; 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. In most cases, the system will add or change the infrastructure that already exists in the organization. The *interface design* specifies how the users will move through the system (e.g., 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 handed to 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.

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). 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. Because the cost of bugs can be immense, testing is one of the most critical steps in implementation. Most organizations give more time and attention to testing than to 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. One of the most important aspects of conversion is the development of a *training plan* 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.

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. Some methodologies are formal standards used by government agencies, whereas others have been developed by consulting firms to sell to clients. Many organizations have internal methodologies that have been honed over the years, and they explain exactly how each phase of the SDLC is to be performed in that company.

There are many ways to categorize methodologies. One way is by looking at whether they focus on business processes or the data that support the business. A *process-centered methodology* emphasizes process models as the core of the system concept. In Figure 1-1, for example, process-centered methodologies would focus first on defining the processes (e.g., assemble sandwich ingredients). *Data-centered methodologies* emphasize data models as the core of the system concept. In Figure 1-1, data-centered methodologies would focus first on defining the contents of the storage areas (e.g., refrigerator) and how the contents were organized.[2] By contrast, *object-oriented methodologies* attempt to balance the focus between process and data by incorporating both into one model. In Figure 1-1, these methodologies would focus first on defining the major elements of the system (e.g., sandwiches, lunches) and look at the processes and data involved with each element.

Another important factor in categorizing methodologies is the sequencing of the SDLC phases and the amount of time and effort devoted to each.[3] In the early days of computing, programmers did not understand the need for formal and well-planned life-cycle methodologies. They tended to move directly from a very simple planning phase right into the construction step of the implementation phase—in other words, from a very fuzzy, not-well-thought-out system request into writing code. This is the same approach that you sometimes use when writing programs for a programming class. It can work for small programs that

[2] The classic modern process-centered methodology is that by Edward Yourdon, *Modern Structured Analysis* (Englewood Cliffs, NJ: Yourdon Press, 1989). An example of a data-centered methodology is information engineering; see James Martin, *Information Engineering,* vols. 1–3 (Englewood Cliffs, NJ: Prentice Hall, 1989). A widely accepted standardized non–object-oriented methodology that balances processes and data is IDEF; see FIPS 183, *Integration Definition for Function Modeling,* Federal Information Processing Standards Publications, U.S. Department of Commerce, 1993.

[3] A good reference for comparing systems development methodologies is Steve McConnell, *Rapid Development* (Redmond, WA: Microsoft Press, 1996).

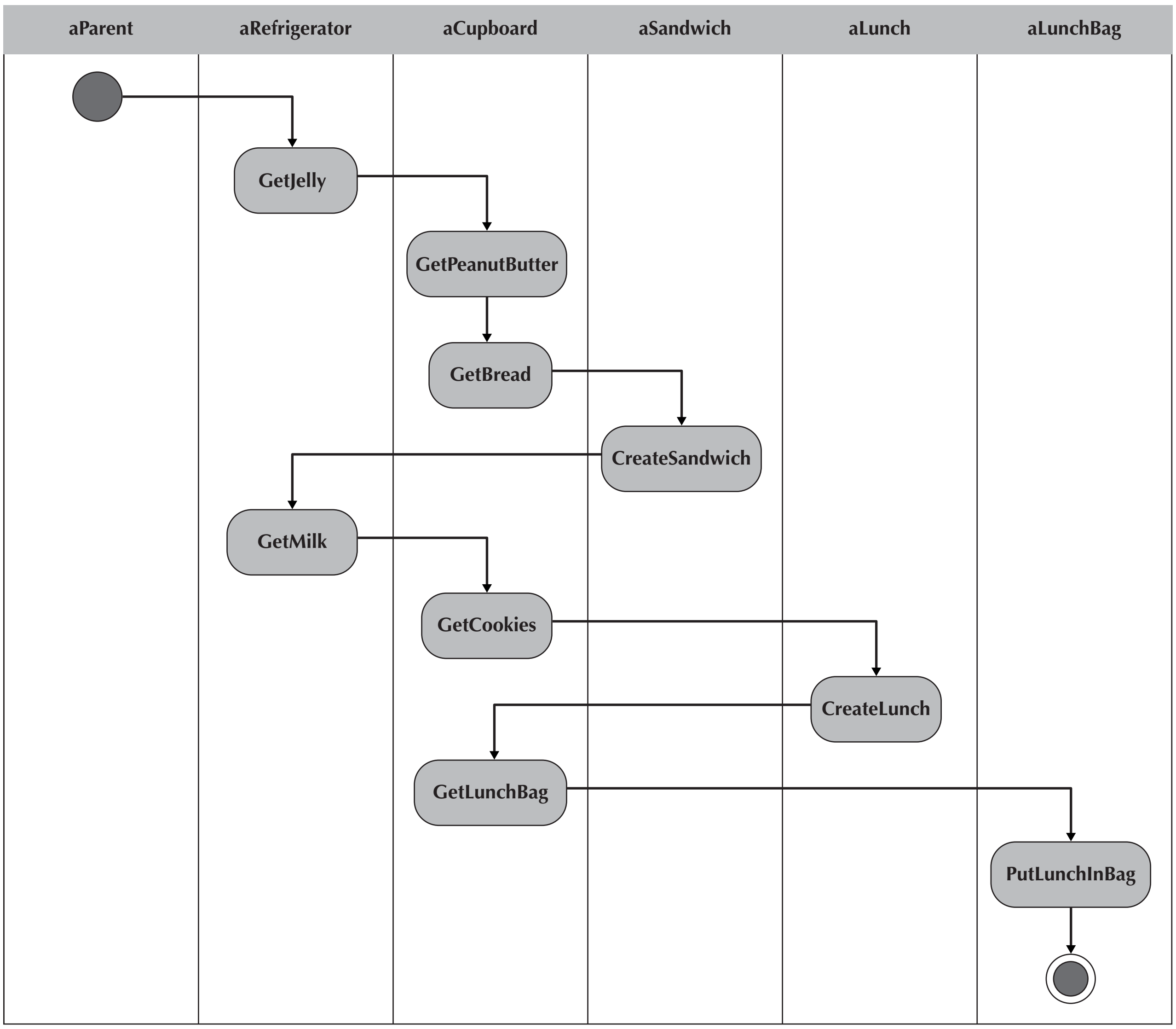


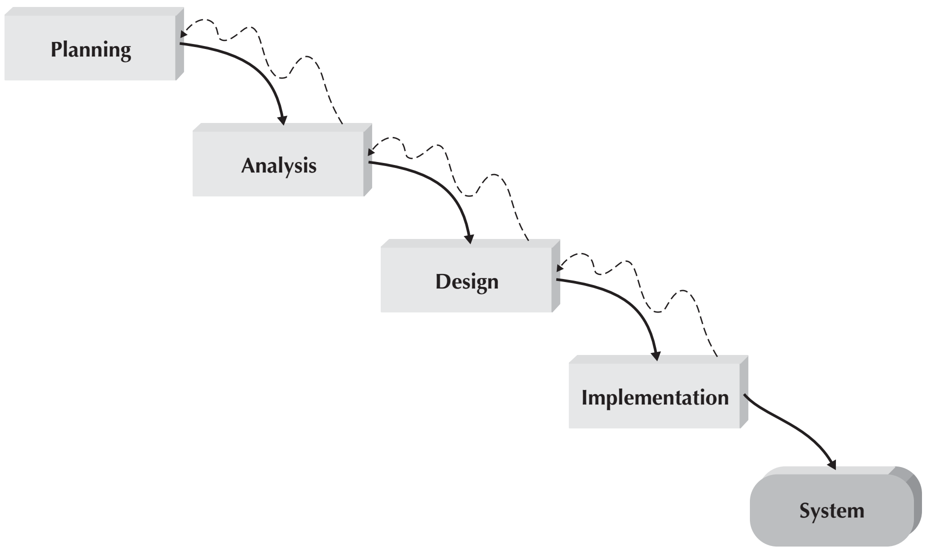
FIGURE 1-1 A Simple Behavioral Model for Making a Simple Lunch

require only one programmer, but if the requirements are complex or unclear, you might miss important aspects of the problem and have to start all over again, throwing away part of the program (and the time and effort spent writing it). This approach also makes teamwork difficult because members have little idea about what needs to be accomplished and how to work together to produce a final product. In this section, we describe three different classes of system development methodologies: structured design, rapid application development, and agile development.

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

FIGURE 1-2
A Waterfall Development-Based Methodology



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.

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. This methodology is referred to as waterfall development because it moves forward from phase to phase in the same manner as a waterfall. Although it is possible to go backward in the SDLC (e.g., from design back to analysis), it is extremely difficult (imagine yourself as a salmon trying to swim upstream against a waterfall, as shown in Figure 1-2).

Structured design also introduced the use of formal modeling or diagramming techniques to describe the basic business processes and the data that support them. Traditional structured design uses one set of diagrams to represent the processes and a separate set of diagrams to represent data. Because two sets of diagrams are used, the systems analyst must decide which set to develop first and use as the core of the system: process-model diagrams or data-model diagrams.

The two key advantages of the structured design waterfall approach are that it identifies system requirements long before programming begins and it minimizes changes to the requirements as the project proceeds. The two key disadvantages are that the design must be completely specified before programming begins and that a long time elapses between the completion of the system proposal in the analysis phase and the delivery of the system (usually many months or years). If the project team misses important requirements, expensive post-implementation programming may be needed (imagine yourself trying to design a car on paper; how likely would you be to remember interior lights that come on when the doors open or to specify the right number of valves on the engine?). A system can also require significant rework because the business environment has changed from the time when the analysis phase occurred.

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, the separate pieces are integrated and the system is delivered (see Figure 1-3).

The primary advantage of this methodology is that it can reduce the time to deliver a system; thus, there is less chance of changes in the business environment causing rework. However, sometimes the subprojects are not completely independent; design decisions made in one subproject can affect another, and the end of the project can require significant integration efforts.

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. In this way, the users can better understand the system and suggest revisions that bring the system closer to what is needed.[4]

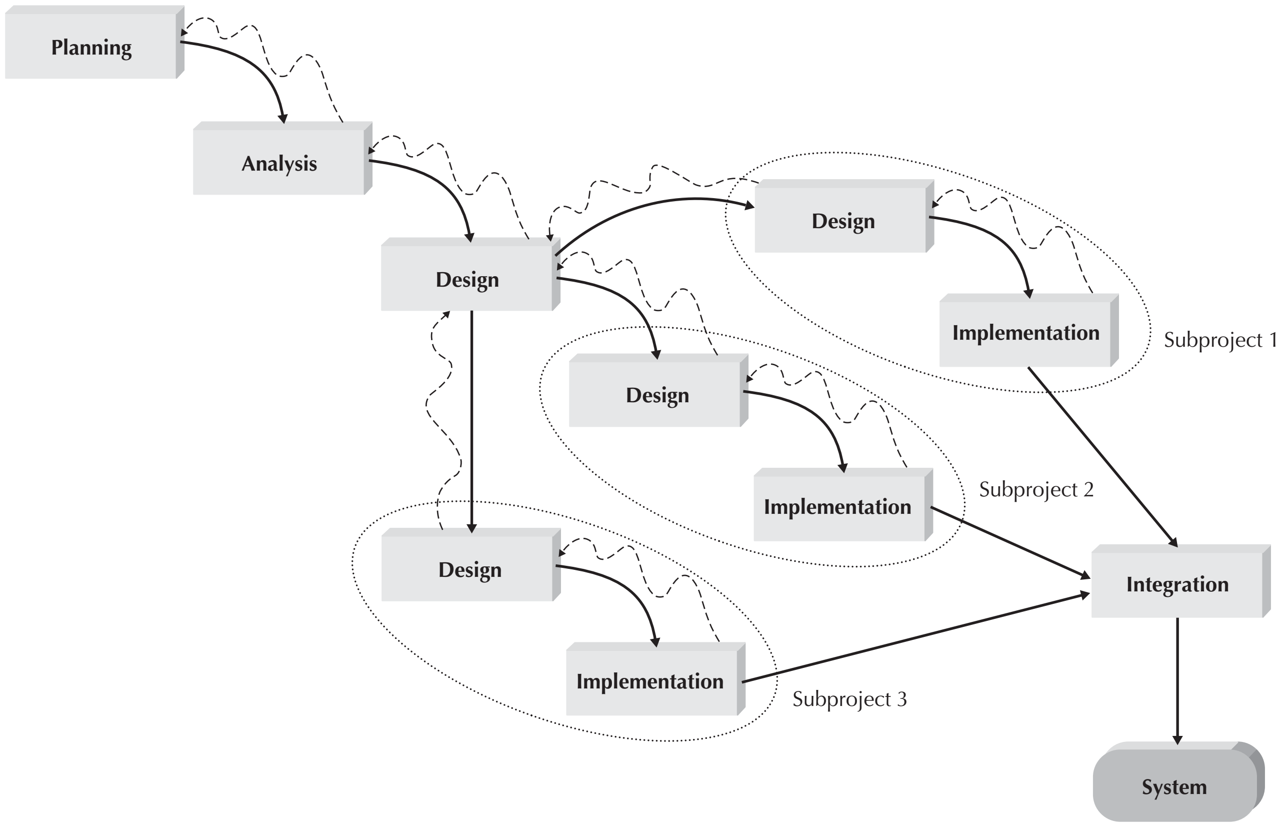


FIGURE 1-3 A Parallel Development-Based Methodology

[4] One of the best RAD books is Steve McConnell, *Rapid Development* (Redmond, WA: Microsoft Press, 1996).

Most RAD-based methodologies recommend that analysts use special techniques and computer tools to speed up the analysis, design, and implementation phases, such as computer-aided software engineering (CASE) tools, joint application design (JAD) sessions, fourth-generation or visual programming languages that simplify and speed up programming, and code generators that automatically produce programs from design specifications. The combination of the changed SDLC phases and the use of these tools and techniques improves the speed and quality of systems development. However, there is one possible subtle problem with RAD-based methodologies: managing user expectations. Owing to the use of the tools and techniques that can improve the speed and quality of systems development, user expectations of what is possible can change dramatically. As a user better understands the information technology (IT), the systems requirements tend to expand. This was less of a problem when using methodologies that spent a lot of time thoroughly documenting requirements.

Phased Development A *phased development*-based methodology breaks an overall system into a series of *versions* that 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).

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.

Phased development-based methodologies have the advantage of quickly getting a useful system into the hands of the users. Although the system does not perform all the functions the users need at first, it does begin to provide business value sooner than if the system were delivered after completion, as is the case with the waterfall and parallel methodologies. Likewise, because users begin to work with the system sooner, they are more likely to identify important additional requirements sooner than with structured design situations.

The major drawback to phased development is that users begin to work with systems that are intentionally incomplete. It is critical to identify the most important and useful features and include them in the first version and to manage users' expectations along the way.

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 quick-and-dirty 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).

The key advantage of a prototyping-based methodology is that it *very* quickly provides a system with which the users can interact, even if it is not ready for widespread organizational use at first. Prototyping reassures the users that the project team is working on the system (there are no long delays in which the users see little progress), and prototyping helps to more quickly refine real requirements.