Modern Systems Analysis and Design

Seventh Edition

Jeffrey A. Hoffer
Joey F. George
Joseph S. Valacich

Chapter 1

The Systems Development Environment

Introduction

- Information Systems Analysis and Design
  - Complex organizational process
  - Used to develop and maintain computer-based information systems
  - Used by a team of business and systems professionals

Introduction (Cont.)

FIGURE 1-1 An organizational approach to systems analysis and design is driven by methodologies, techniques, and tools

Introduction (Cont.)

- Application Software
  - Computer software designed to support organizational functions or processes
- Systems Analyst
  - Organizational role most responsible for analysis and design of information systems

A Modern Approach to Systems Analysis and Design

- 1950s: focus on efficient automation of existing processes
- 1960s: advent of procedural third generation languages (3GL) faster and more reliable computers
- 1970s: system development becomes more like an engineering discipline

A Modern Approach to Systems Analysis and Design (Cont.)

- 1980s: major breakthrough with 4GL, CASE tools, object-oriented methods
- 1990s: focus on system integration, GUI applications, client/server platforms, Internet
- The new century: Web application development, wireless PDAs and smart phones, component-based applications, application service providers (ASP)

Developing Information Systems

- **System Development Methodology** is a standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems.

Systems Development Life Cycle (SDLC)

- Traditional methodology used to develop, maintain, and replace information systems
- Phases in SDLC:
  - Planning
  - Analysis
  - Design
  - Implementation
  - Maintenance

Standard and Evolutionary Views of SDLC

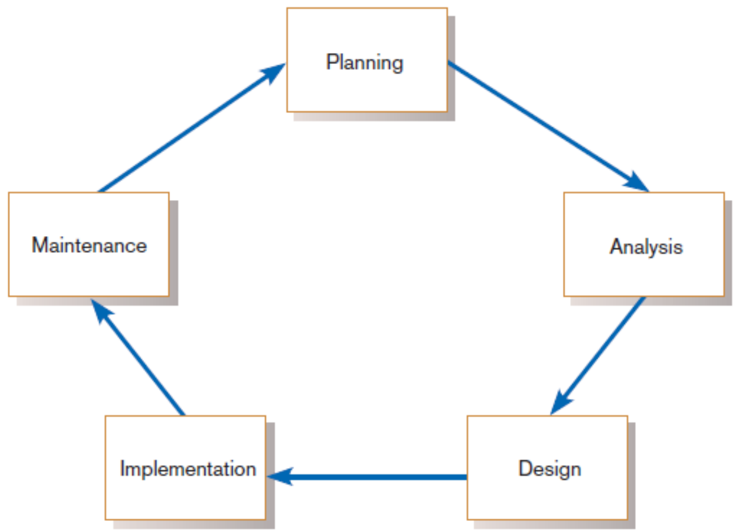


FIGURE 1-2
Systems development life cycle

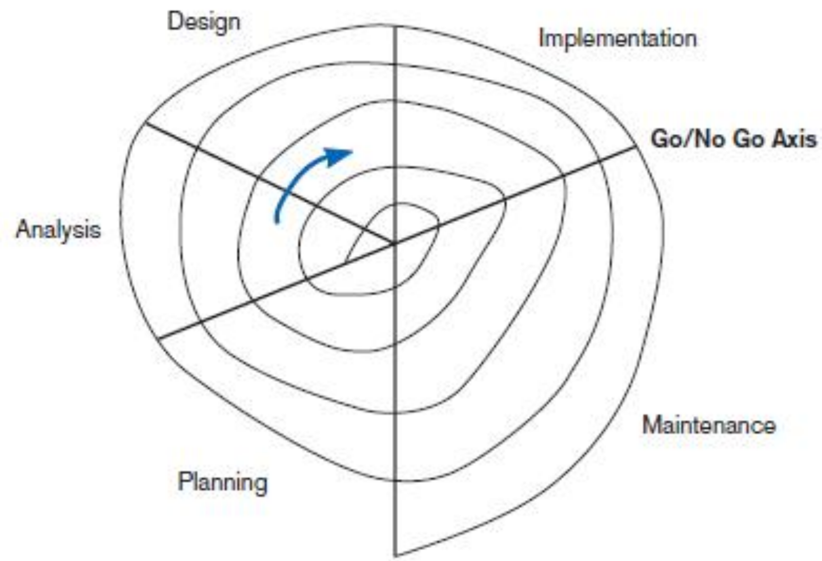


FIGURE 1-3 Evolutionary model

Systems Development Life Cycle (SDLC) (Cont.)

- **Planning** – an organization's total information system needs are identified, analyzed, prioritized, and arranged
- **Analysis** – system requirements are studied and structured
- **Design** – a description of the recommended solution is converted into logical and then physical system specifications

Systems Development Life Cycle (SDLC) (Cont.)

- **Logical design** – all functional features of the system chosen for development in analysis are described independently of any computer platform
- **Physical design** – the logical specifications of the system from logical design are transformed into the technology-specific details from which all programming and system construction can be accomplished

Systems Development Life Cycle (SDLC) (Cont.)

- **Implementation** – the information system is coded, tested, installed and supported in the organization
- **Maintenance** – an information system is systematically repaired and improved

TABLE 1-1 Products of SDLC Phases

Phase	Products, Outputs, or Deliverables
Planning	Priorities for systems and projects; an architecture for data, networks, and selection hardware, and information systems management are the result of associated systems
	Detailed steps, or work plan, for project
	Specification of system scope and planning and high-level system requirements or features
	Assignment of team members and other resources
	System justification or business case
Analysis	Description of current system and where problems or opportunities are with a general recommendation on how to fix, enhance, or replace current system
	Explanation of alternative systems and justification for chosen alternative
Design	Functional, detailed specifications of all system elements (data, processes, inputs, and outputs)
	Technical, detailed specifications of all system elements (programs, files, network, system software, etc.)
	Acquisition plan for new technology
Implementation	Code, documentation, training procedures, and support capabilities
Maintenance	New versions or releases of software with associated updates to documentation, training, and support

The Heart of the Systems Development Process

FIGURE 1-8
Analysis–design–code–test loop

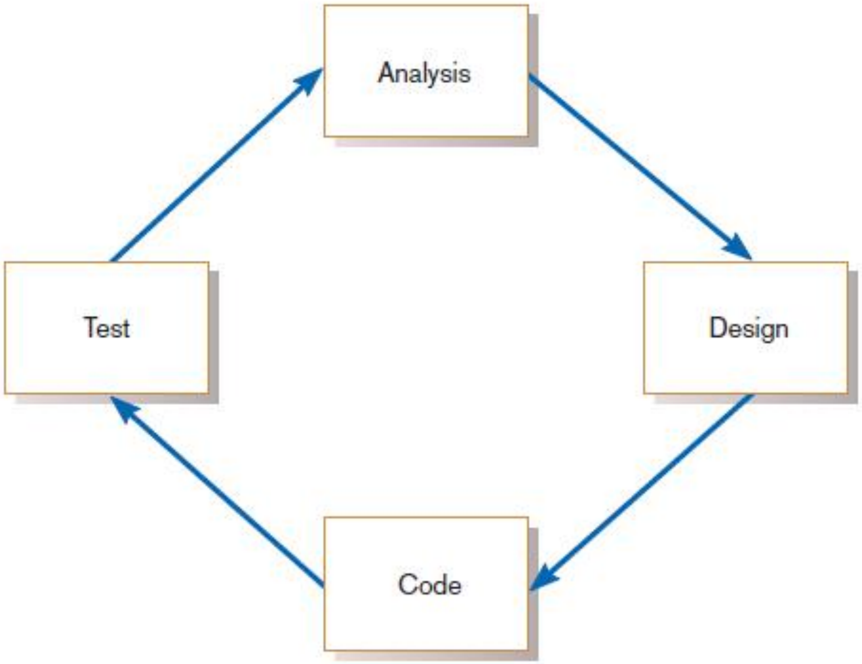
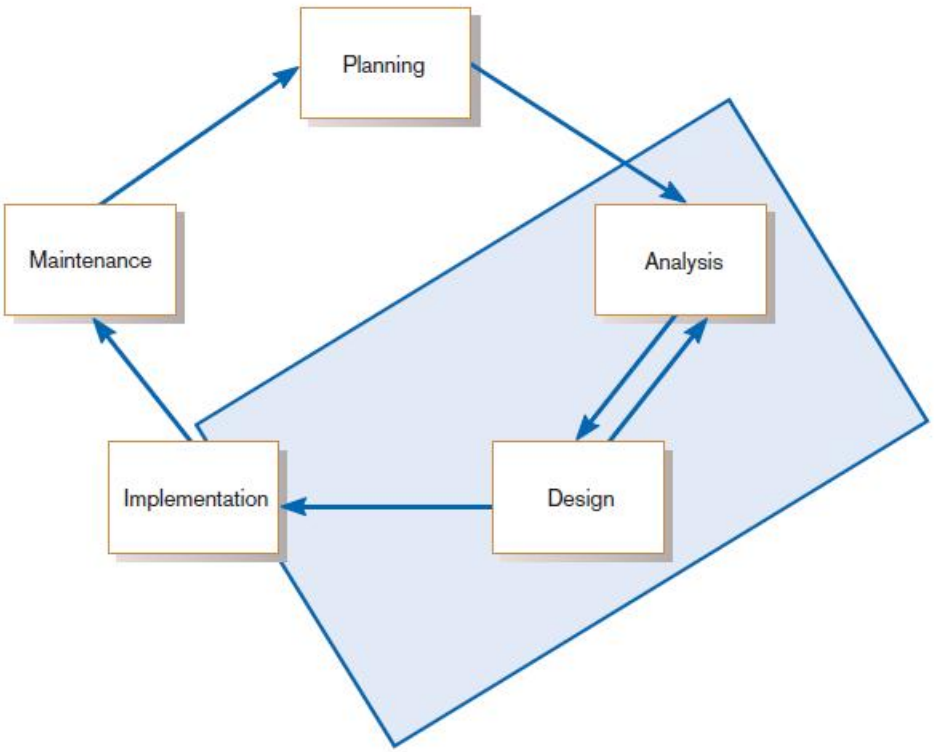


FIGURE 1-9
The heart of systems development



Current practice combines analysis, design, and implementation into a single iterative and parallel process of activities.

Traditional Waterfall SDLC

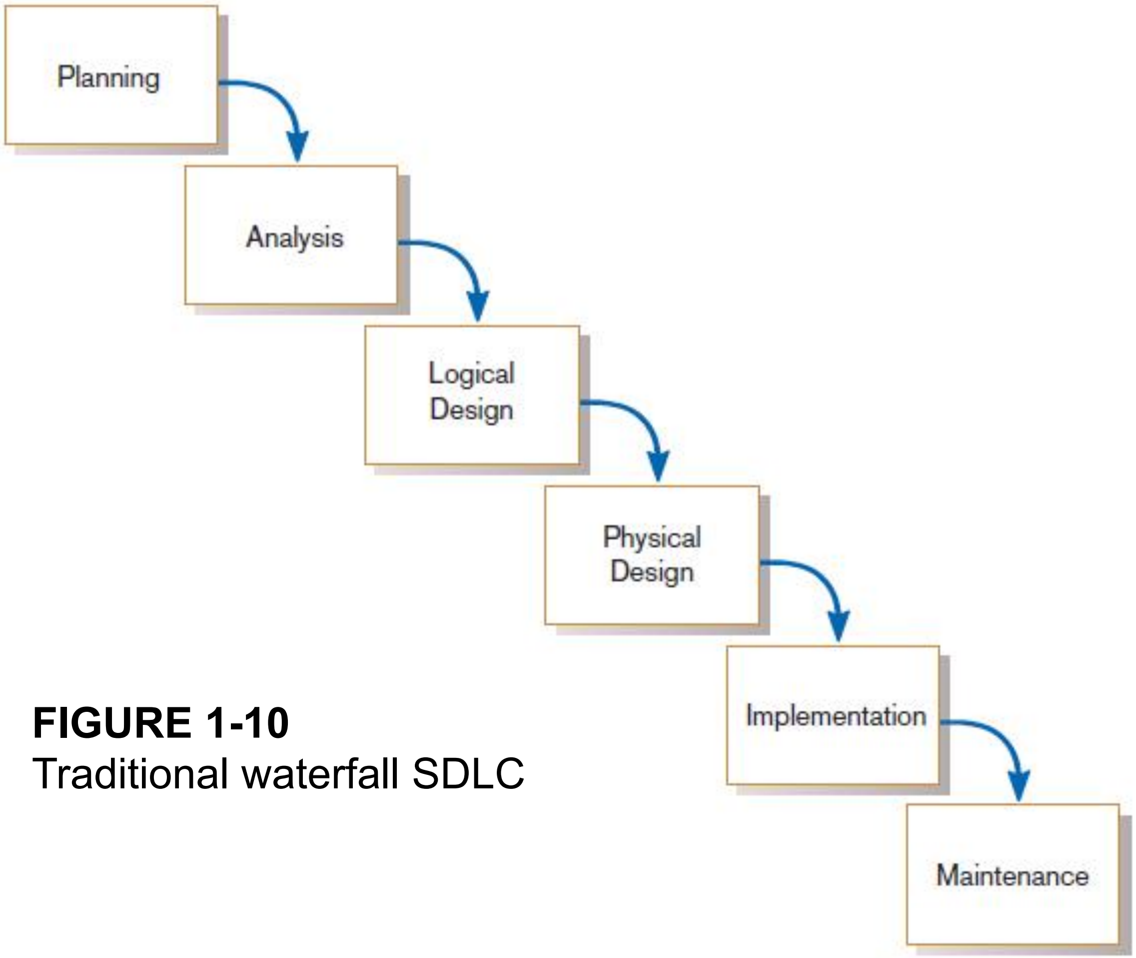


FIGURE 1-10
Traditional waterfall SDLC

One phase begins when another completes, with little backtracking and looping.

Problems with Waterfall Approach

- Feedback ignored, milestones lock in design specs even when conditions change
- Limited user involvement (only in requirements phase)
- Too much focus on milestone deadlines of SDLC phases to the detriment of sound development practices

Different Approaches to Improving Development

- CASE Tools
- Rapid Application Development (RAD)
- Agile Methodologies
- eXtreme Programming

Computer-Aided Software Engineering (CASE) Tools

- Diagramming tools enable graphical representation.
- Computer displays and report generators help prototype how systems “look and feel”.
- IBM’s Rational products are the best known CASE tools.

Computer-Aided Software Engineering (CASE) Tools (Cont.)

- Analysis tools automatically check for consistency in diagrams, forms, and reports.
- A central repository provides integrated storage of diagrams, reports, and project management specifications.

Computer-Aided Software Engineering (CASE) Tools (Cont.)

- Documentation generators standardize technical and user documentation.
- Code generators enable automatic generation of programs and database code directly from design documents, diagrams, forms, and reports.

CASE Tools (Cont.)

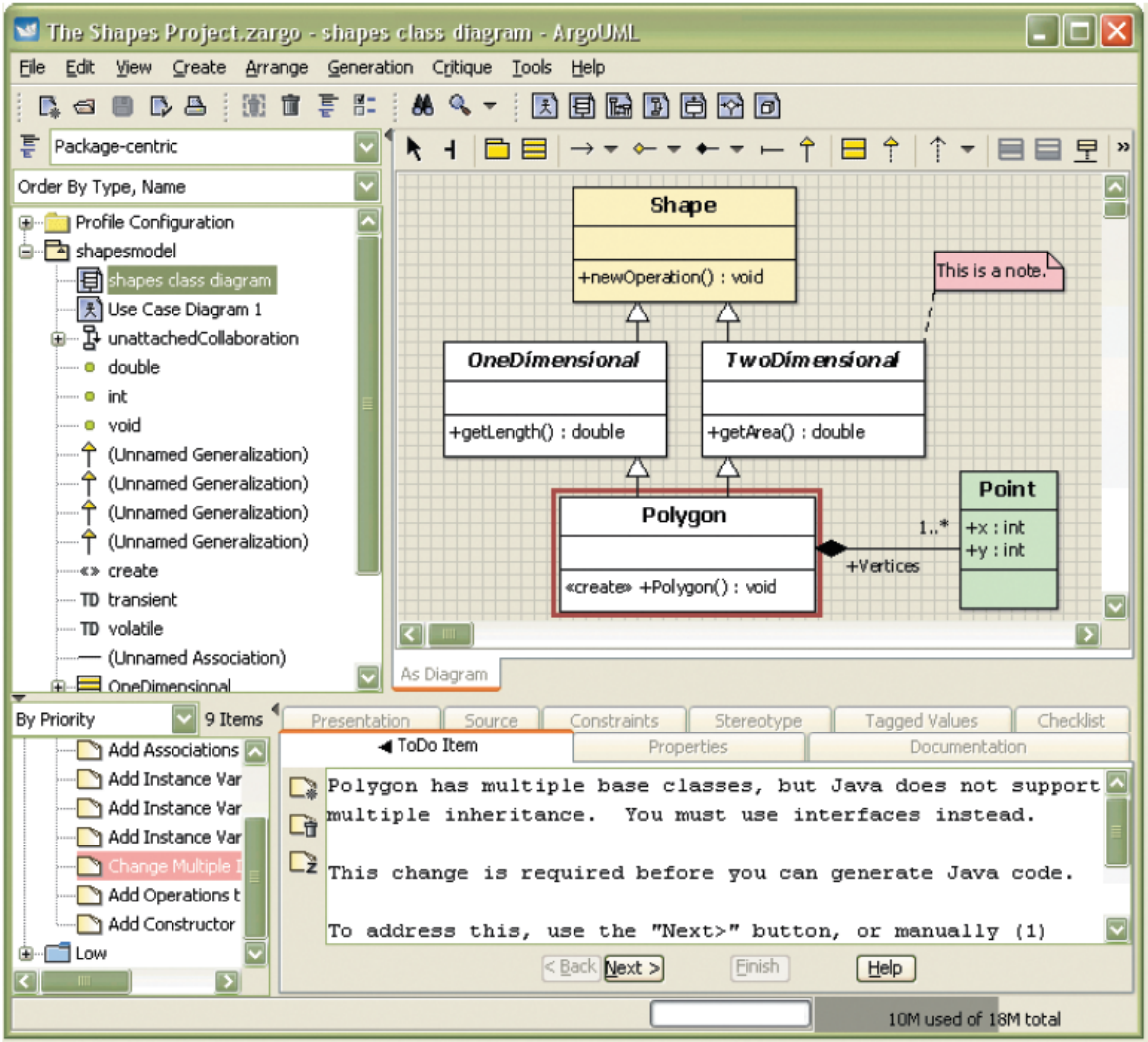


FIGURE 1-11
Screen shot of ArgoUML, an open source CASE tool

(*Source: http://argouml.tigris.org/*)

CASE Tools (Cont.)

TABLE 1-2 Examples of CASE Usage within the SDLC

SDLC Phase	Key Activities	CASE Tool Usage
Project identification and selection	Display and structure high-level organizational information	Diagramming and matrix tools to create and structure information
Project initiation and planning	Develop project scope and feasibility	Repository and documentation generators to develop project plans
Analysis	Determine and structure system requirements	Diagramming to create process, logic, and data models
Logical and physical design	Create new system designs	Form and report generators to prototype designs; analysis and documentation generators to define specifications
Implementation	Translate designs into an information system	Code generators and analysis, form and report generators to develop system; documentation generators to develop system and user documentation
Maintenance	Evolve information system	All tools are used (repeat life cycle)

Rapid Application Development (RAD)

- Decreases design and implementation time
- Involves: extensive user involvement, prototyping, integrated CASE tools, code generators
- More focus on user interface and system function, less on detailed business analysis and system performance

Rapid Application Development (RAD) (Cont.)

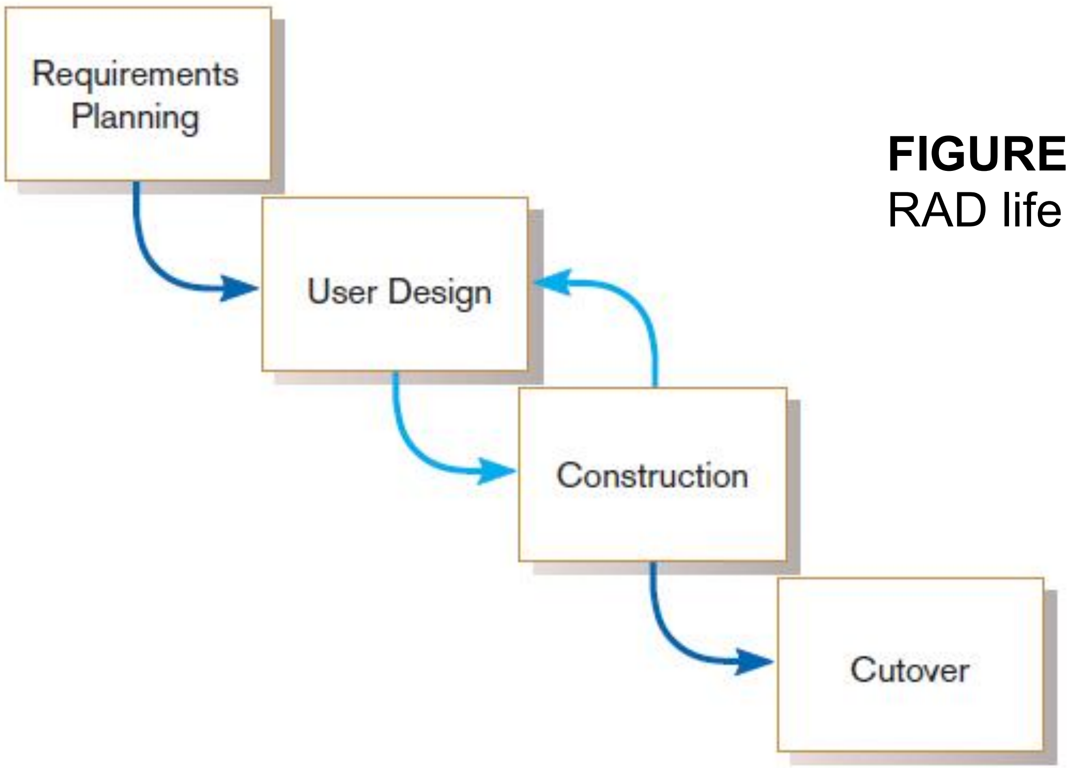


FIGURE 1-12
RAD life cycle

Agile Methodologies

- Motivated by recognition of software development as fluid, unpredictable, and dynamic
- Three key principles
  - Adaptive rather than predictive
  - Emphasize people rather than roles
  - Self-adaptive processes

TABLE 1-3 The Agile Manifesto

The Manifesto for Agile Software Development

Seventeen anarchists agree:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- *Individuals and interactions* over processes and tools.
- *Working software* over comprehensive documentation.
- *Customer collaboration* over contract negotiation.
- *Responding to change* over following a plan.

That is, while we value the items on the right, we value the items on the left more. We follow the following principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Businesspeople and developers work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

—Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas (*www .agileAlliance.org*)

The Agile Methodologies group argues that software development methodologies adapted from engineering generally do not fit with real-world software development.

When to use Agile Methodologies

- If your project involves:
  - Unpredictable or dynamic requirements
  - Responsible and motivated developers
  - Customers who understand the process and will get involved

service

TABLE 1-4 Five Critical Factors That Distinguish Agile and Traditional Approaches to Systems Development

Factor	Agile Methods	Traditional Methods
Size	Well matched to small products and teams. Reliance on tacit knowledge limits scalability.	Methods evolved to handle large products and teams. Hard to tailor down to small projects.
Criticality	Untested on safety-critical products. Potential difficulties with simple design and lack of documentation.	Methods evolved to handle highly critical products. Hard to tailor down to products that are not critical.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments but a source of potentially expensive rework for highly stable environments.	Detailed plans and Big Design Up Front, excellent for highly stable environment but a source of expensive rework for highly dynamic environments.
Personnel	Requires continuous presence of a critical mass of scarce experts. Risky to use no-agile people.	Needs a critical mass of scarce experts during project definition but can work with fewer later in the project, unless the environment is highly dynamic.
Culture	Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom (thriving on chaos).	Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear practices and procedures (thriving on order).

(*Source:* Boehm, Barry; Turner, Richard, Balancing Agility and Discipline: A Guide for the Perplexed, 1st Ed., (c)2004. Reprinted and electronically reproduced by permission of Pearson Education, Inc. Upper Saddle River, New Jersey.

eXtreme Programming

- Short, incremental development cycles
- Automated tests
- Two-person programming teams
- Coding, testing, listening, designing

eXtreme Programming (Cont.)

- Coding and testing operate together
- Advantages:
  - Communication between developers
  - High level of productivity
  - High-quality code

Object-Oriented Analysis and Design (OOAD)

- Based on objects rather than data or processes
- **Object**: a structure encapsulating attributes and behaviors of a real-world entity

Object-Oriented Analysis and Design (OOAD) (Cont.)

- **Object class**: a logical grouping of objects sharing the same attributes and behaviors
- **Inheritance**: hierarchical arrangement of classes enable subclasses to inherit properties of superclasses

Rational Unified Process (RUP)

- An object-oriented systems development methodology
- Establishes four phase of development: inception, elaboration, construction, and transition
  - Each phase is organized into a number of separate iterations.

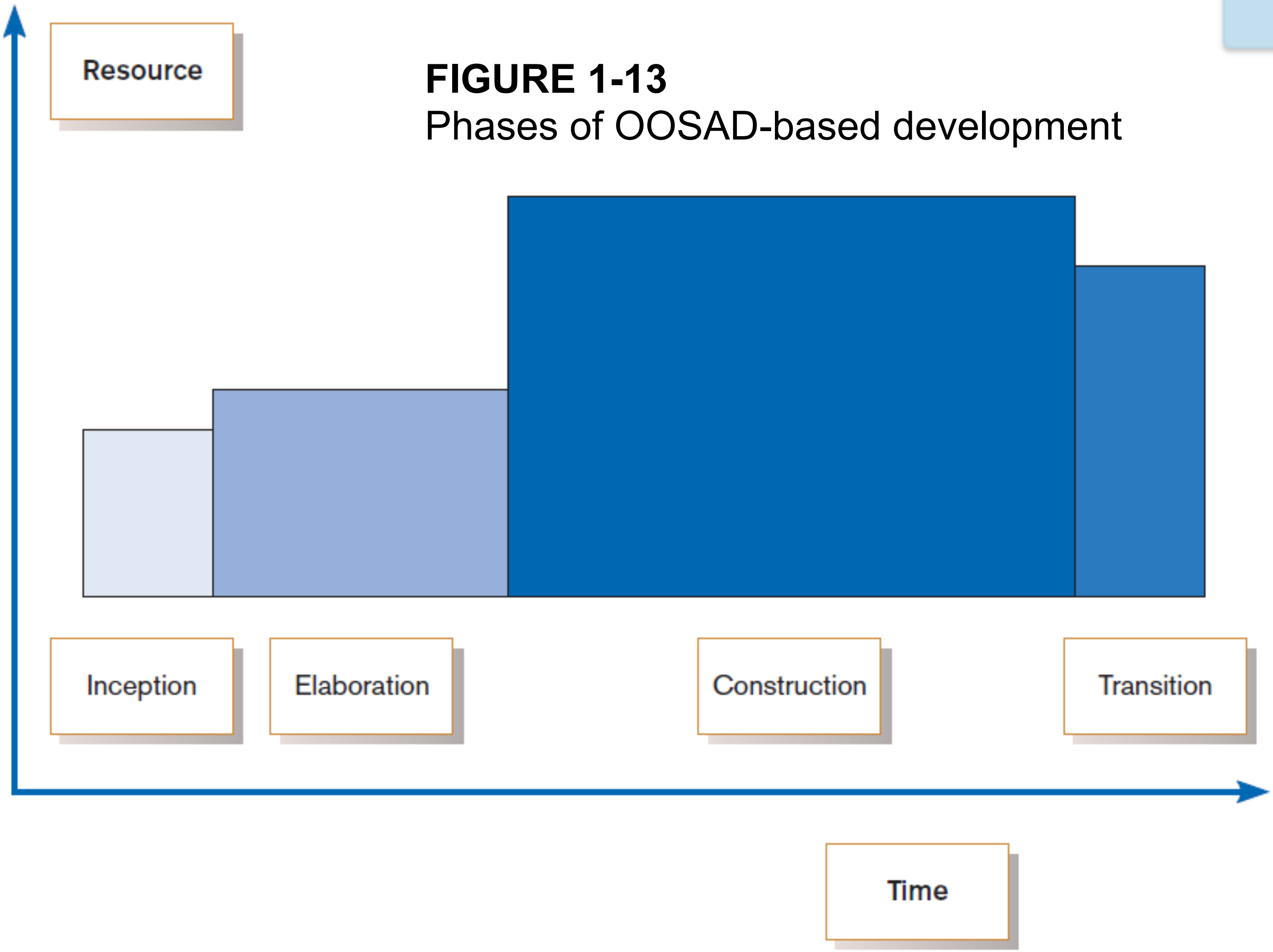


FIGURE 1-13
Phases of OOSAD-based development

Our Approach to Systems Development

- The SDLC is an organizing and guiding principle in this book.
- We may construct artificial boundaries or artificially separate activities and processes for learning purposes.
- Our intent is to help you understand all the pieces and how to assemble them.