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

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

FIGURE 1-10
Traditional waterfall SDLC
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
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/)
## TABLE 1-2  Examples of CASE Usage within the SDLC

<table>
<thead>
<tr>
<th>SDLC Phase</th>
<th>Key Activities</th>
<th>CASE Tool Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project identification and selection</td>
<td>Display and structure high-level organizational information</td>
<td>Diagramming and matrix tools to create and structure information</td>
</tr>
<tr>
<td>Project initiation and planning</td>
<td>Develop project scope and feasibility</td>
<td>Repository and documentation generators to develop project plans</td>
</tr>
<tr>
<td>Analysis</td>
<td>Determine and structure system requirements</td>
<td>Diagramming to create process, logic, and data models</td>
</tr>
<tr>
<td>Logical and physical design</td>
<td>Create new system designs</td>
<td>Form and report generators to prototype designs; analysis and documentation generators to define specifications</td>
</tr>
<tr>
<td>Implementation</td>
<td>Translate designs into an information system</td>
<td>Code generators and analysis, form and report generators to develop system; documentation generators to develop system and user documentation</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Evolve information system</td>
<td>All tools are used (repeat life cycle)</td>
</tr>
</tbody>
</table>
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
The Agile Methodologies group argues that software development methodologies adapted from engineering generally do not fit with real-world software development.

**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]

(Source: http://agilemanifesto.org/ © 2001, the above authors this declaration may be freely copied in any form, but only in its entirety through this notice.)
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
<table>
<thead>
<tr>
<th>Factor</th>
<th>Agile Methods</th>
<th>Traditional Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Well matched to small products and teams. Reliance on tacit knowledge limits scalability.</td>
<td>Methods evolved to handle large products and teams. Hard to tailor down to small projects.</td>
</tr>
<tr>
<td>Criticality</td>
<td>Untested on safety-critical products. Potential difficulties with simple design and lack of documentation.</td>
<td>Methods evolved to handle highly critical products. Hard to tailor down to products that are not critical.</td>
</tr>
<tr>
<td>Dynamism</td>
<td>Simple design and continuous refactoring are excellent for highly dynamic environments but a source of potentially expensive rework for highly stable environments.</td>
<td>Detailed plans and Big Design Up Front, excellent for highly stable environment but a source of expensive rework for highly dynamic environments.</td>
</tr>
<tr>
<td>Personnel</td>
<td>Requires continuous presence of a critical mass of scarce experts. Risky to use no-agile people.</td>
<td>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.</td>
</tr>
<tr>
<td>Culture</td>
<td>Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom (thrive on chaos).</td>
<td>Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear practices and procedures (thrive on order).</td>
</tr>
</tbody>
</table>

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

Inception  Elaboration  Construction  Transition

Time

Resource
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.
This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.