Essentials of Systems Analysis and Design Sixth Edition

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

Appendix B
Agile Methodologies

Learning Objectives

- ✓ Define **Agile** Methodologies
- ✓ Explain when to use Agile Methodologies and when to use engineering-based approaches to systems development
- ✓ Define eXtreme Programming
- ✓ Discuss the Agile Methodology approach to systems requirements determination, design specifications, and the combination of coding and testing

The Trend to Agile Methodologies

- Three Phases of Systems Analysis Since Inception
  - Undisciplined (Developer as Artist)
    - Lack of documentation and development tools
    - High degree of dependence upon developer for maintenance
  - Developer as Engineer
    - Documentation
    - Rigorous testing
    - Structured tools and techniques
    - Computer-based tools

The Trend to Agile Methodologies (continued)

- Three Phases of Systems Analysis Since Inception (continued)
  - Agile Methodologies
    - Object-oriented approach
    - Sacrifices milestones and multiple phases
    - Close cooperation between developers and clients
    - Many life cycle phases combined into one
    - Multiple rapid releases of software

Agile Methodologies

- Many Different Methodologies under Umbrella
  - Crystal family of methodologies
  - Adaptive Software Development
  - Scrum
  - Feature Driven Development
  - eXtreme Programming
- Agile Manifesto Developed in February 2001 (Figure B-1)
  - Three Key Principles:
    - Focus on adaptive rather than predictive methodologies
    - Focus on people rather than roles
    - Self-adaptive process

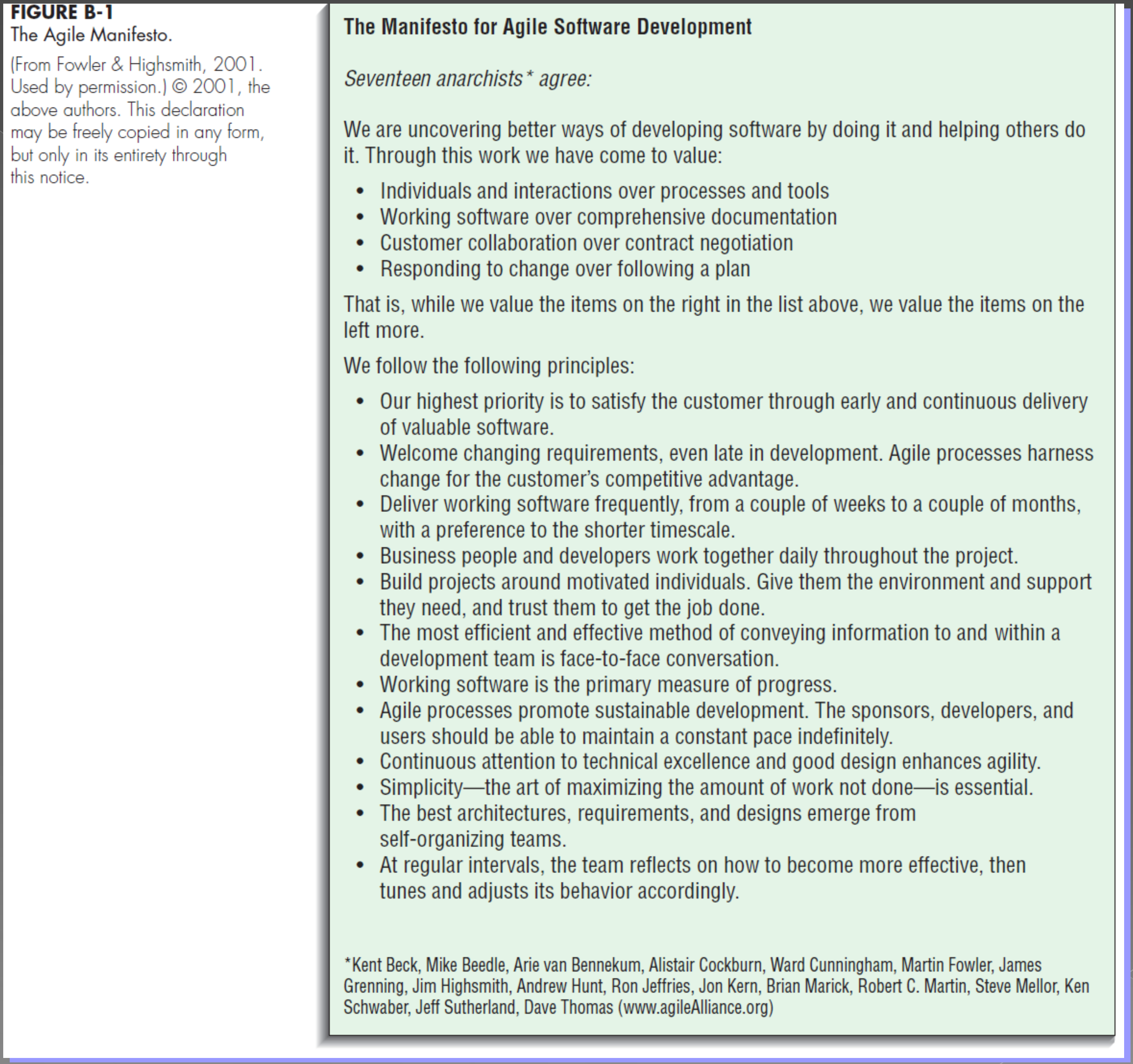

FIGURE B-1
The Agile Manifesto.

(From Fowler & Highsmith, 2001. Used by permission.) © 2001, the above authors. This declaration may be freely copied in any form, but only in its entirety through this notice.

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 in the list above, 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.
- Business people 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.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- 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)

Agile Methodologies(continued)

- Engineering Methodologies Do Not Fit Well with Software Development
  - Engineering requirements are well understood; software requirements often are not
  - In many engineering projects, design is only 15 percent of project; in software engineering, design can be as much as 50 percent
- Agile Methodologies Embrace Change and Deal with Lack of Predictability
  - Iterative development
    - Provides feedback to customers and developers alike
  - Focus on individuals
  - Adaptive software development process
    - Process can be refined and improved as software is developed
- Agile Methodologies are Recommended for Projects That Involve:
  - Unpredictable or dynamic requirements
  - Responsible and motivated developers
  - Customers who understand and will get involved
- Engineering Approach is Recommended when:
  - Project involves more than 100 people
  - Project is operating under fixed-price or fixed-scope contract

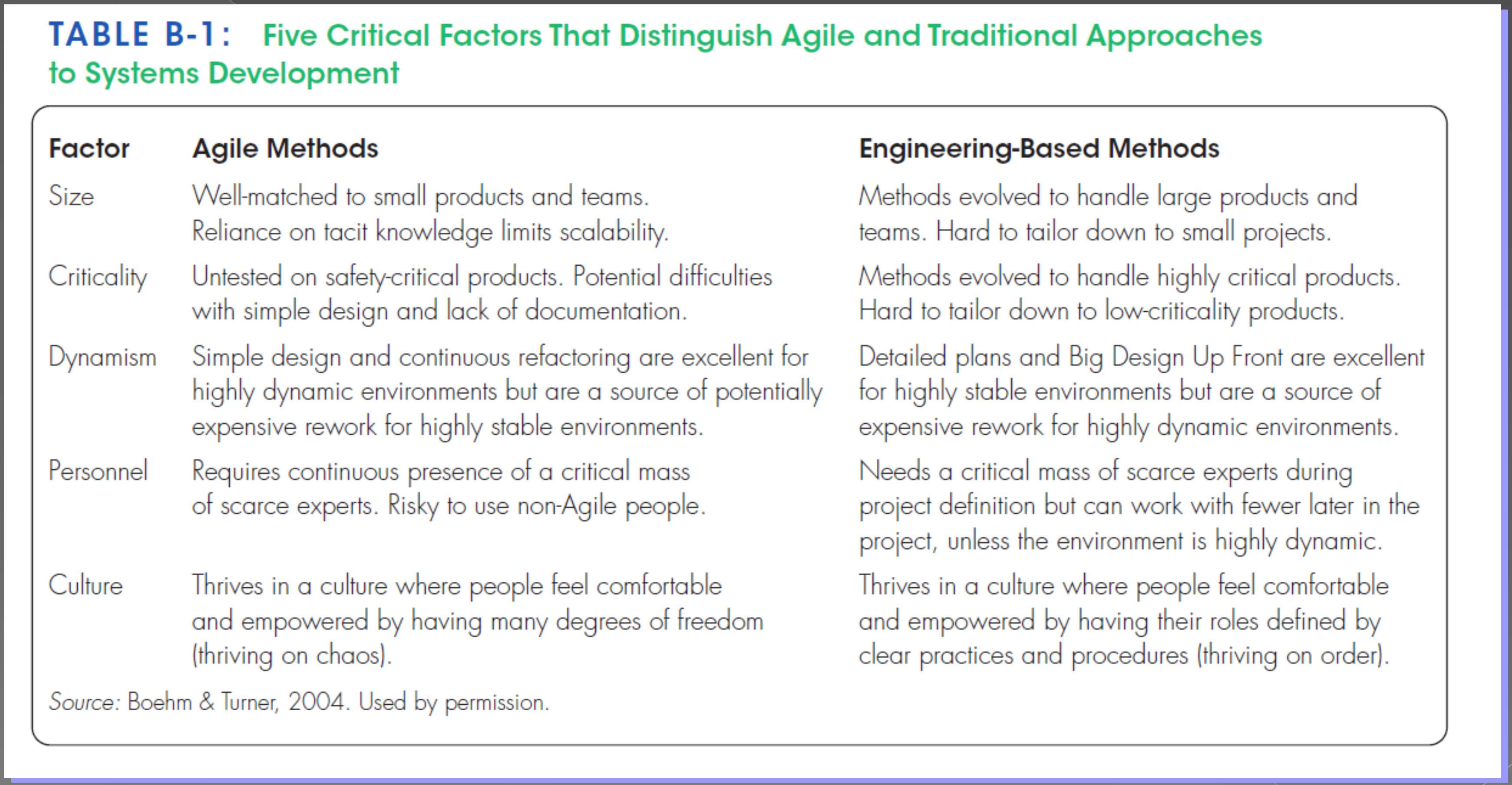

TABLE B-1: Five Critical Factors That Distinguish Agile and Traditional Approaches to Systems Development

Factor	Agile Methods	Engineering-Based 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 low-criticality products.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments but are a source of potentially expensive rework for highly stable environments.	Detailed plans and Big Design Up Front are excellent for highly stable environments but are a source of expensive rework for highly dynamic environments.
Personnel	Requires continuous presence of a critical mass of scarce experts. Risky to use non-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 & Turner, 2004. Used by permission.

eXtreme Programming

- One Example of Agile Methodologies
- Distinguished by:
  - Short development cycles
  - Incremental planning approach
  - Focus on automated tests
  - Reliance on evolutionary approach to development
  - Use of two-person (pair) programming teams
  - Customer on-site during development process
- Relates to Design Specifications
  - Planning, design, analysis, and construction are fused together into a single phase of activity
  - Systems requirements are captured and presented in a unique way
  - Programmers who write code also write the tests
- Pair Programming
  - All programming is done by two people working together
  - Includes coding and testing
  - Advantages
    - More communication among developers
    - Higher levels of productivity
    - Higher quality code
    - Reinforcement of other practices, such as code and test discipline

Requirements Determination

- Continual User Involvement
  - Traditional Approach
    - Users involved in early stage
    - User signs off at project completion
    - Often, system business processes changed during development
      - System many times did not adequately address user's needs
  - Iterative Process of User Feedback
- Agile Usage-Centered Design
  - Process of eight steps
  - See Table B-2
  - Most effective steps:
    - Venting session
    - Use of 3 x 5 cards

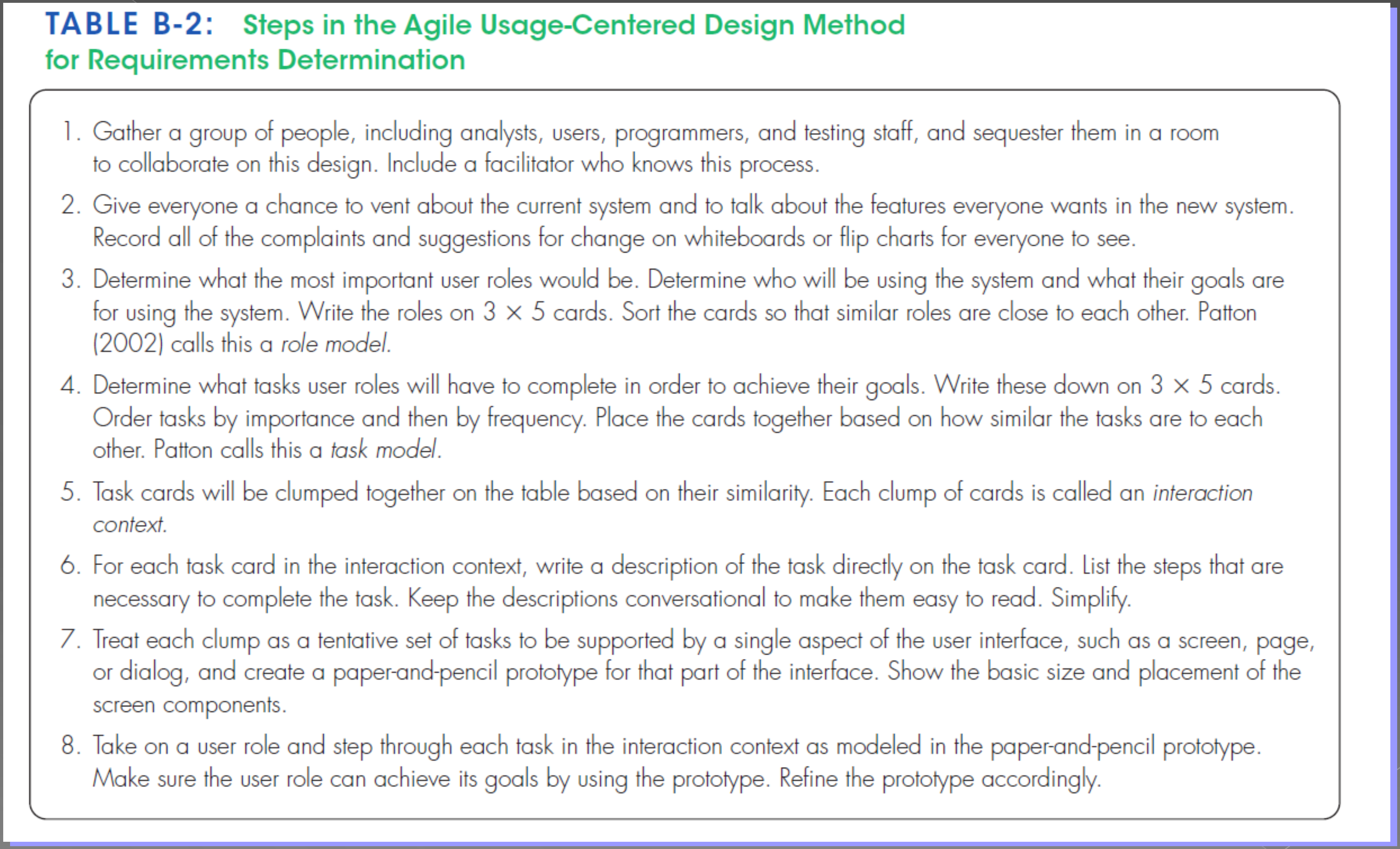

TABLE B-2: Steps in the Agile Usage-Centered Design Method for Requirements Determination

1. Gather a group of people, including analysts, users, programmers, and testing staff, and sequester them in a room to collaborate on this design. Include a facilitator who knows this process.
2. Give everyone a chance to vent about the current system and to talk about the features everyone wants in the new system. Record all of the complaints and suggestions for change on whiteboards or flip charts for everyone to see.
3. Determine what the most important user roles would be. Determine who will be using the system and what their goals are for using the system. Write the roles on 3 × 5 cards. Sort the cards so that similar roles are close to each other. Patton (2002) calls this a *role model.*
4. Determine what tasks user roles will have to complete in order to achieve their goals. Write these down on 3 × 5 cards. Order tasks by importance and then by frequency. Place the cards together based on how similar the tasks are to each other. Patton calls this a *task model*.
5. Task cards will be clumped together on the table based on their similarity. Each clump of cards is called an *interaction context.*
6. For each task card in the interaction context, write a description of the task directly on the task card. List the steps that are necessary to complete the task. Keep the descriptions conversational to make them easy to read. Simplify.
7. Treat each clump as a tentative set of tasks to be supported by a single aspect of the user interface, such as a screen, page, or dialog, and create a paper-and-pencil prototype for that part of the interface. Show the basic size and placement of the screen components.
8. Take on a user role and step through each task in the interaction context as modeled in the paper-and-pencil prototype. Make sure the user role can achieve its goals by using the prototype. Refine the prototype accordingly.

Requirements Determination (continued)

- The Planning Game
  - Players
    - Business (Customers)
    - Development
  - Game pieces are story cards
    - Created by Business
    - Contain description of a feature or procedure
    - Each card is dated and numbered
  - Three Phases
    - Exploration
      - Business creates a story card
      - Development responds with a time estimate
    - Commitment
      - Business sorts cards into three stacks: essential, non-essential but add value, and nice to have
      - Development sorts story cards according to risk
      - Business selects cards that will be used in next release
    - Steering
      - Business sees how development is progressing
      - Plan is adjusted accordingly
      - Can take place as often as once every three weeks

Requirements Determination (continued)

- The Planning Game (continued)
  - Iteration Planning Game
    - Follows planning game
    - Played only by programmers
    - Uses task cards
    - Same three phases
      - Exploration
        - Programmers convert story cards to task cards
      - Commitment
        - Programmers accept responsibility for tasks
        - Balance workloads
      - Steering
        - Write code and test
        - Integrate into product
    - Takes place between steering phase meetings of planning game

Design Specifications

- eXtreme Programming Employs Two Techniques to Continually Improve Quality of Design
  - Simple Design
    - Four constraints
      - The system must communicate everything that you want it to communicate
      - The system must contain no duplicate code
      - The system should have the fewest possible classes
      - The system should have the fewest possible methods
  - Refactoring
    - Process of simplifying a system after new features have been added

B.14

Implementation

- Traditional systems development involves an implementation phase
- In eXtreme Programming and Other Agile Methodologies, implementation is complete once testing has been done

B.15

Summary

- Trend to Agile Methodologies
- eXtreme Programming as an Example
  - Continual user involvement
  - Agile-Centered approach
  - Pair programming
  - The Planning Game

B.16