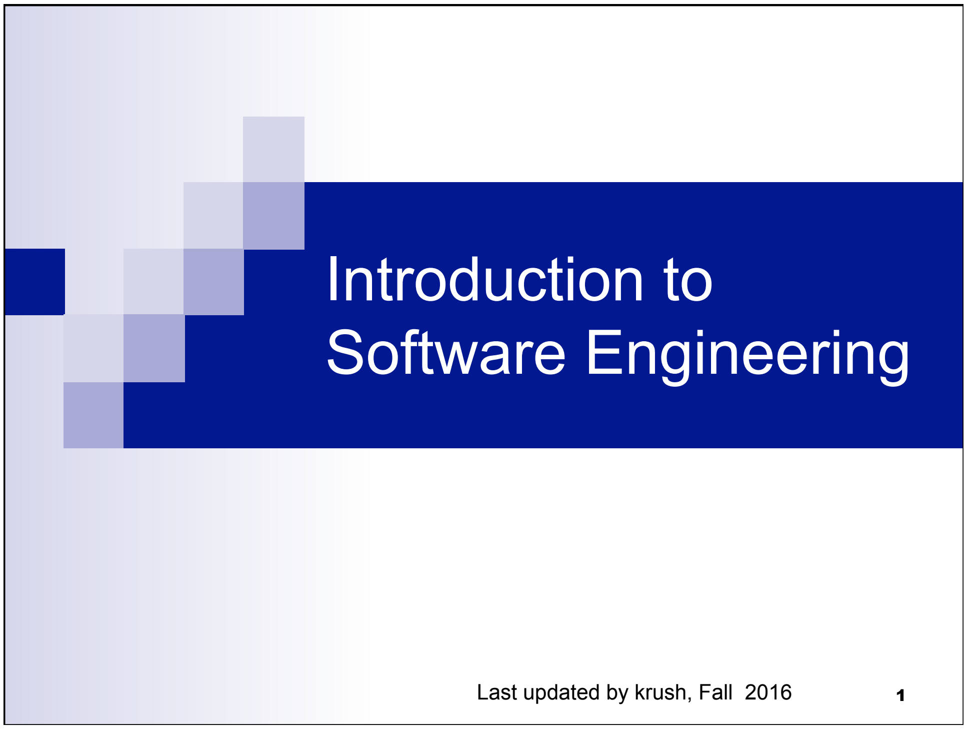
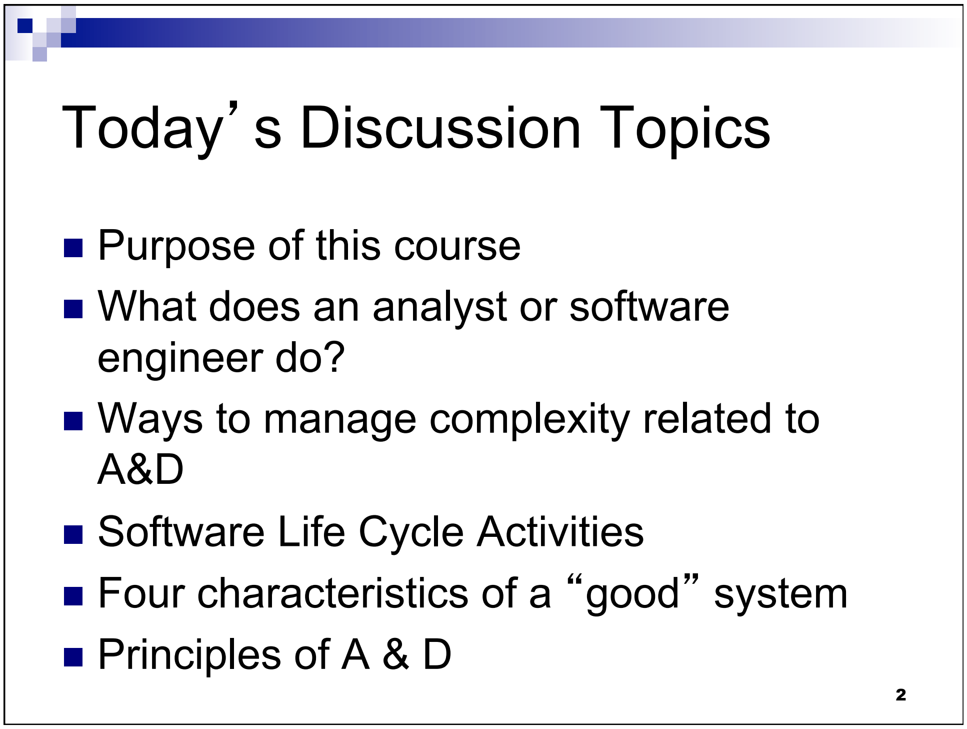


Introduction to Software Engineering

Last updated by krush, Fall 2016

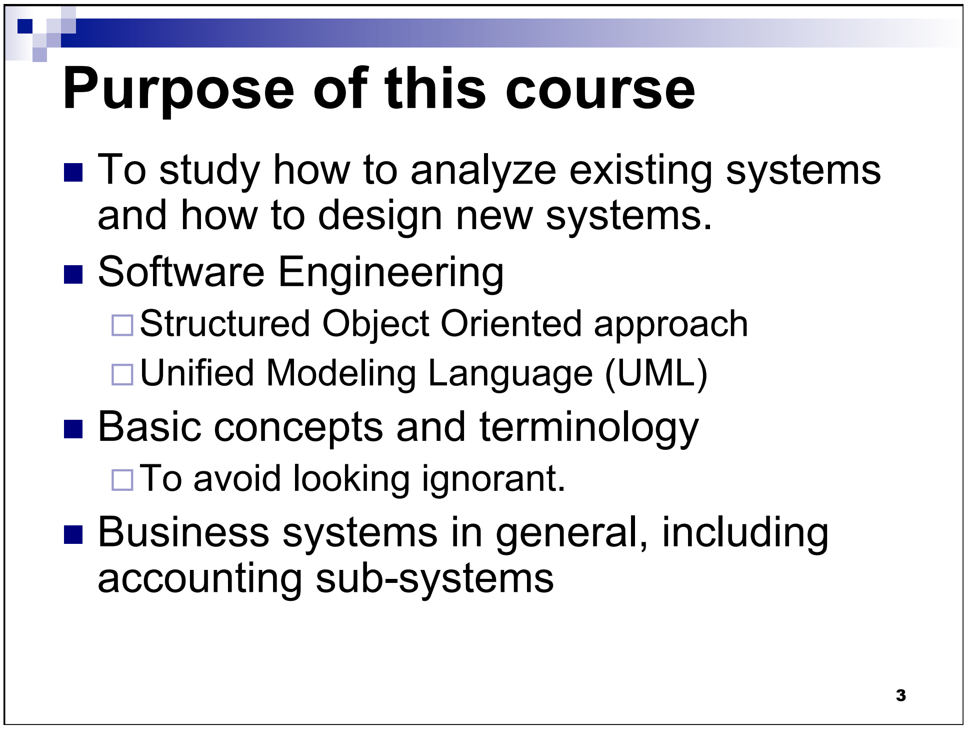
1



Today’s Discussion Topics

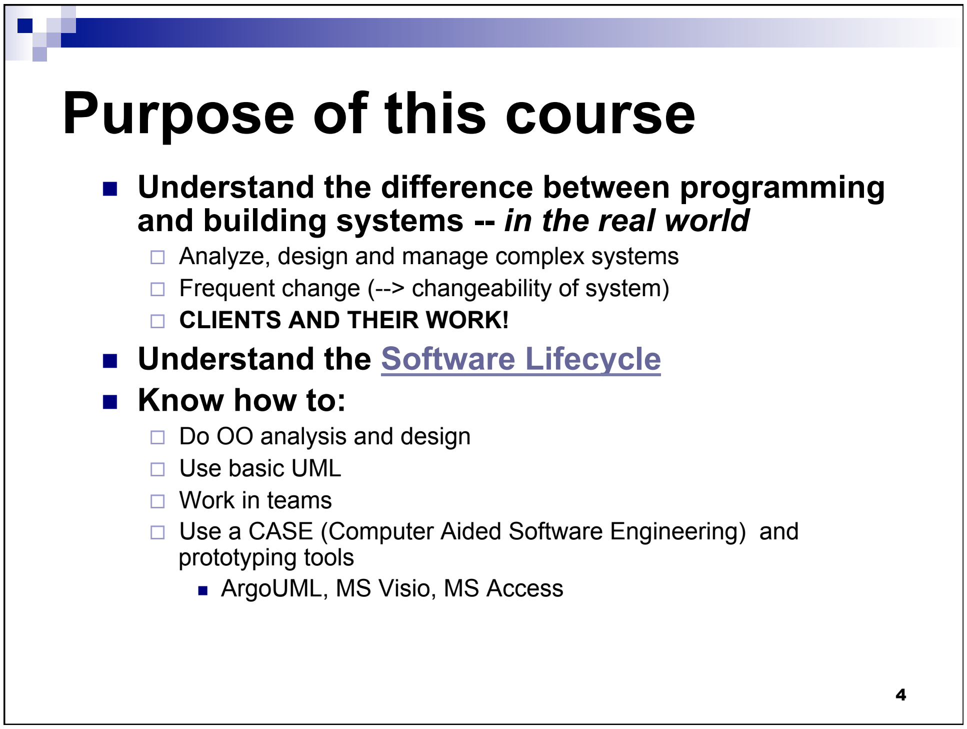
- Purpose of this course
- What does an analyst or software engineer do?
- Ways to manage complexity related to A&D
- Software Life Cycle Activities
- Four characteristics of a “good” system
- Principles of A & D

2



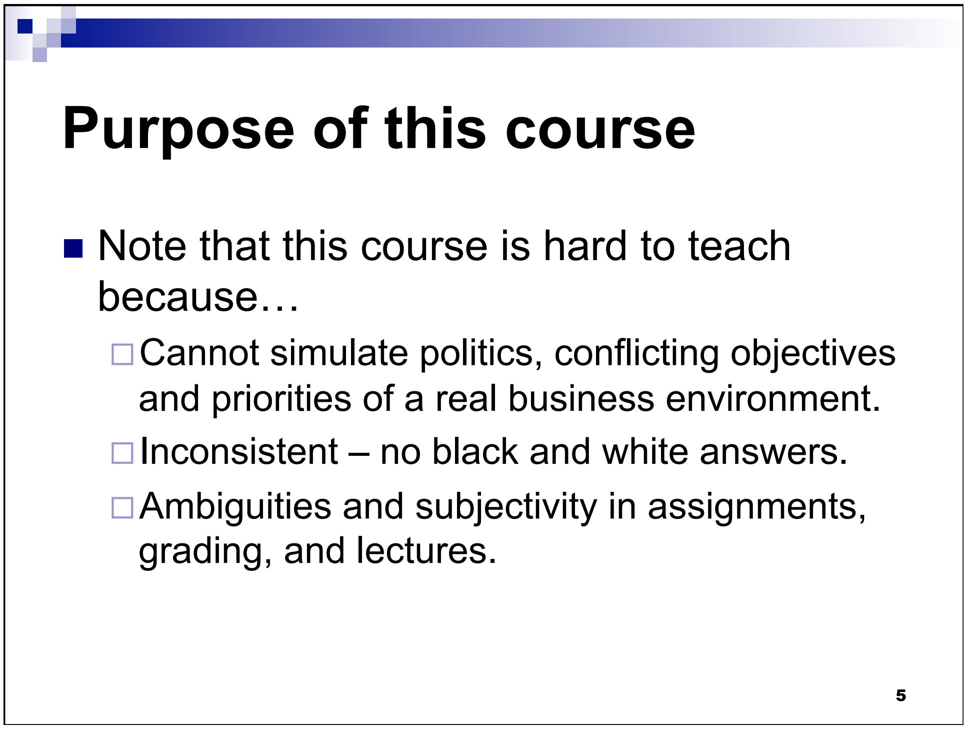
Purpose of this course

- To study how to analyze existing systems and how to design new systems.
- Software Engineering
  - □ Structured Object Oriented approach
  - □ Unified Modeling Language (UML)
- Basic concepts and terminology
  - □ To avoid looking ignorant.
- Business systems in general, including accounting sub-systems



Purpose of this course

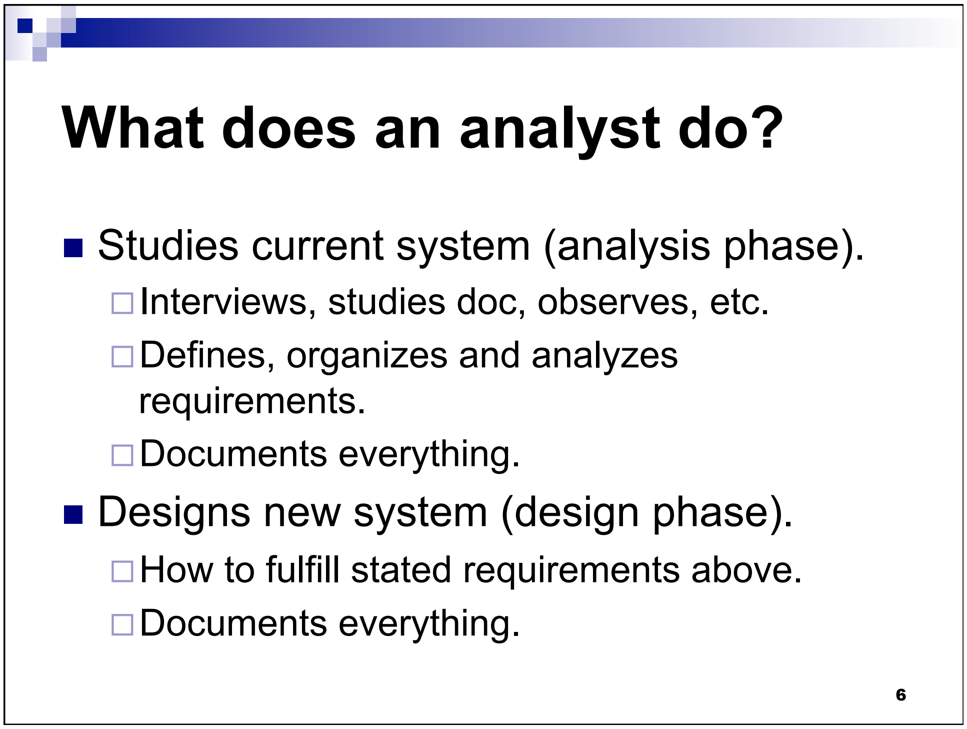
- **Understand the difference between programming and building systems -- *in the real world***
  - □ Analyze, design and manage complex systems
  - □ Frequent change (--> changeability of system)
  - □ **CLIENTS AND THEIR WORK!**
- **Understand the Software Lifecycle**
- **Know how to:**
  - □ Do OO analysis and design
  - □ Use basic UML
  - □ Work in teams
  - □ Use a CASE (Computer Aided Software Engineering) and prototyping tools
    - ArgoUML, MS Visio, MS Access



Purpose of this course

- Note that this course is hard to teach because…
  - □ Cannot simulate politics, conflicting objectives and priorities of a real business environment.
  - □ Inconsistent – no black and white answers.
  - □ Ambiguities and subjectivity in assignments, grading, and lectures.

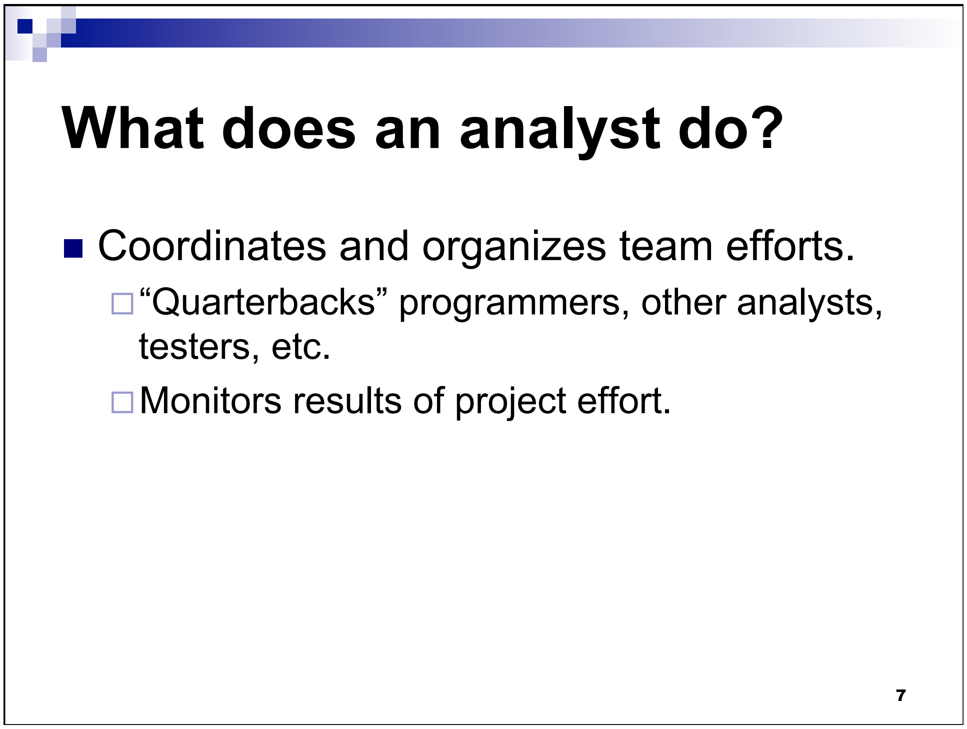
5



What does an analyst do?

- Studies current system (analysis phase).
  - □ Interviews, studies doc, observes, etc.
  - □ Defines, organizes and analyzes requirements.
  - □ Documents everything.
- Designs new system (design phase).
  - □ How to fulfill stated requirements above.
  - □ Documents everything.

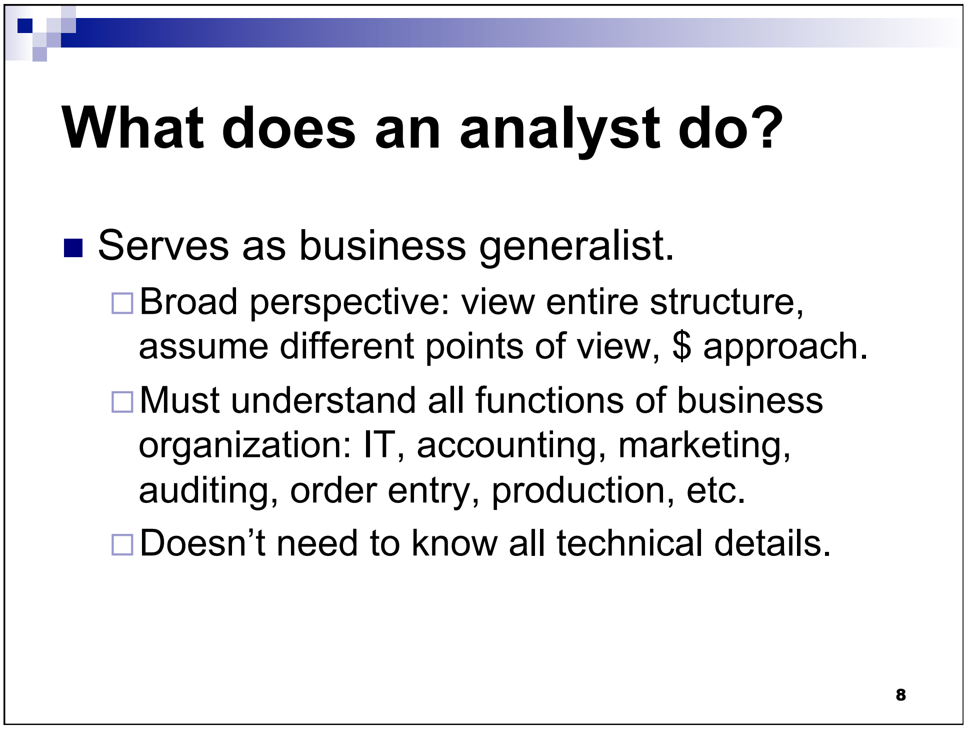
6



What does an analyst do?

- Coordinates and organizes team efforts.
  - “Quarterbacks” programmers, other analysts, testers, etc.
  - Monitors results of project effort.

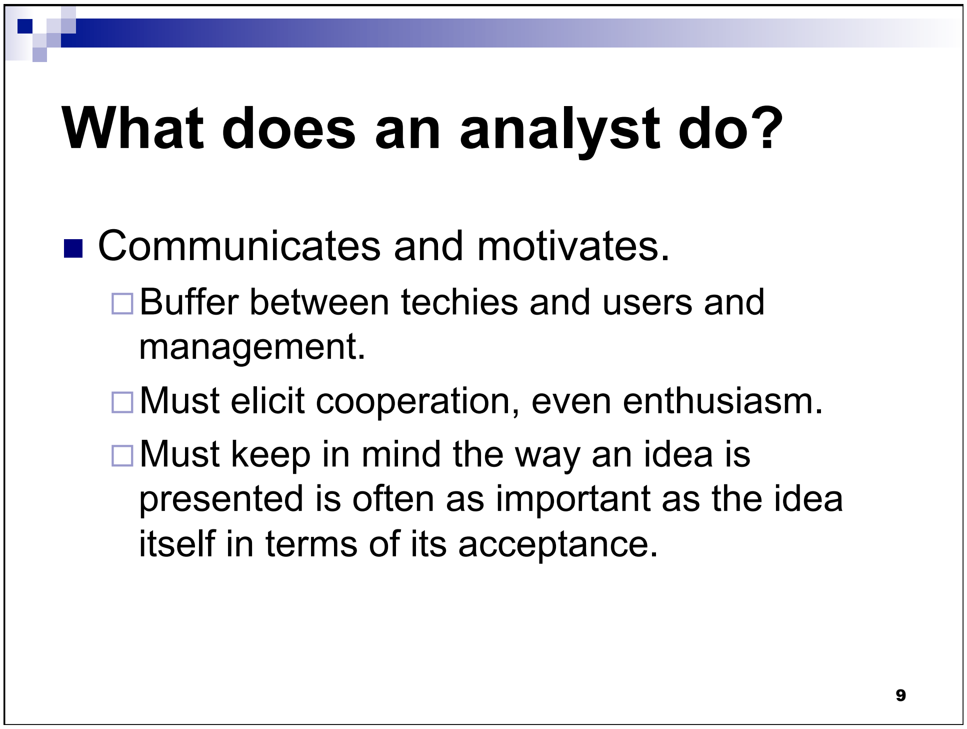
7



What does an analyst do?

- Serves as business generalist.
  - Broad perspective: view entire structure, assume different points of view, $ approach.
  - Must understand all functions of business organization: IT, accounting, marketing, auditing, order entry, production, etc.
  - Doesn’t need to know all technical details.

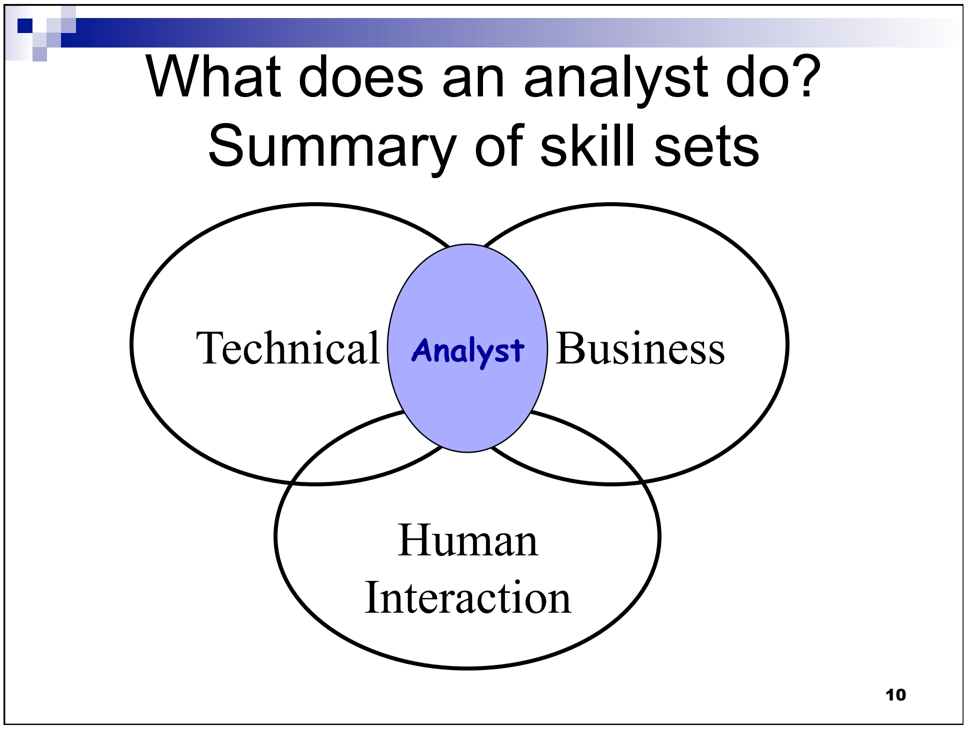
8



What does an analyst do?

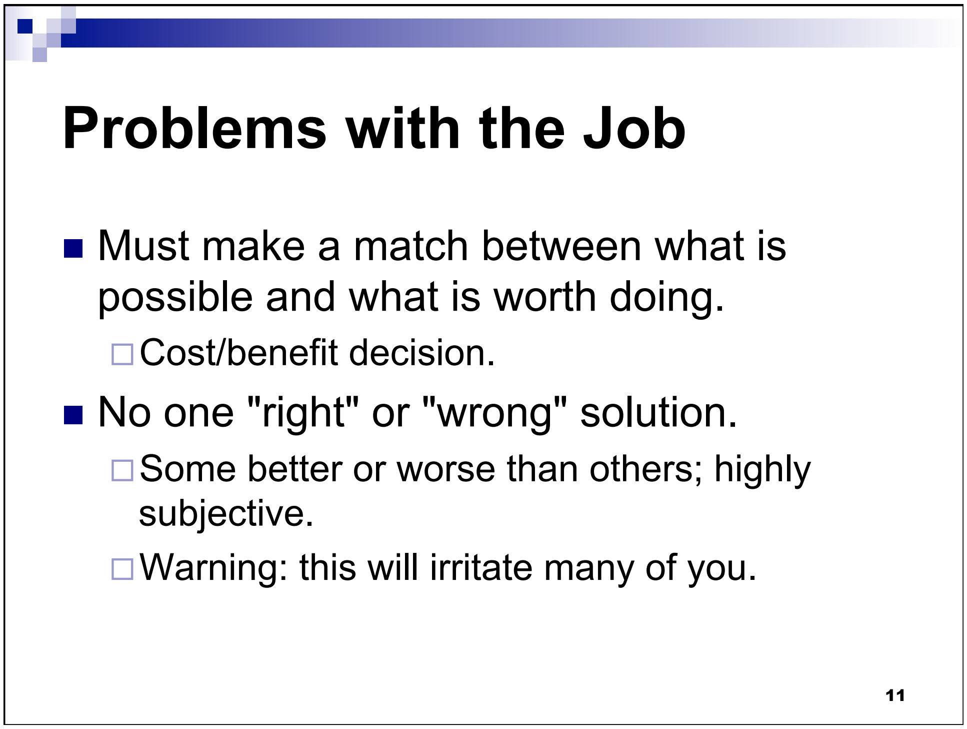
- Communicates and motivates.
  - □ Buffer between techies and users and management.
  - □ Must elicit cooperation, even enthusiasm.
  - □ Must keep in mind the way an idea is presented is often as important as the idea itself in terms of its acceptance.

9



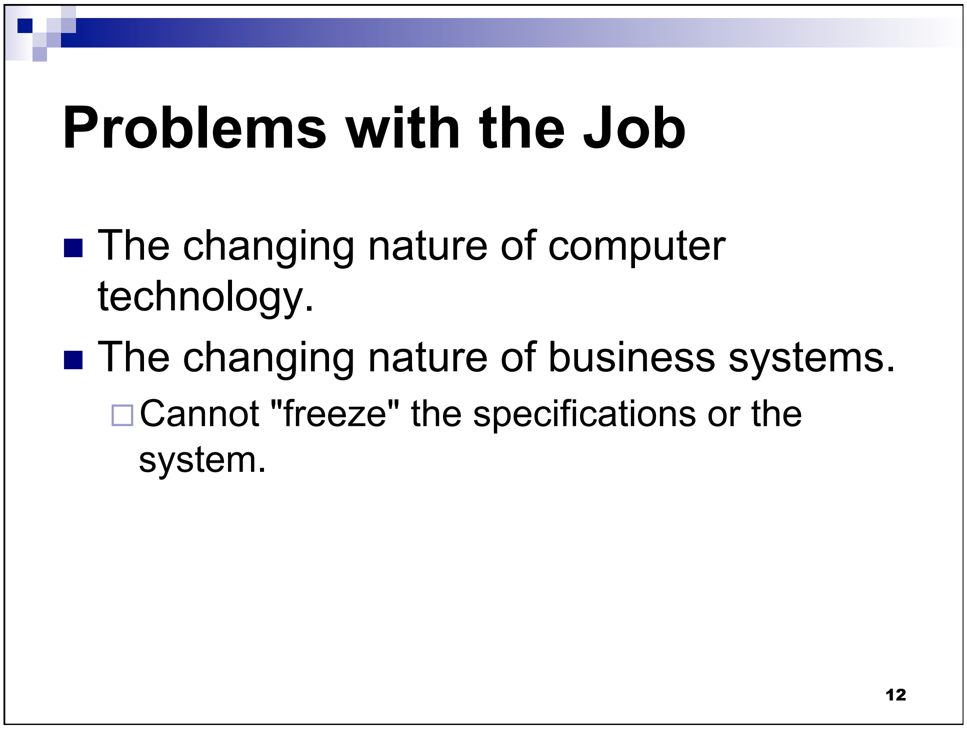
What does an analyst do? Summary of skill sets

10



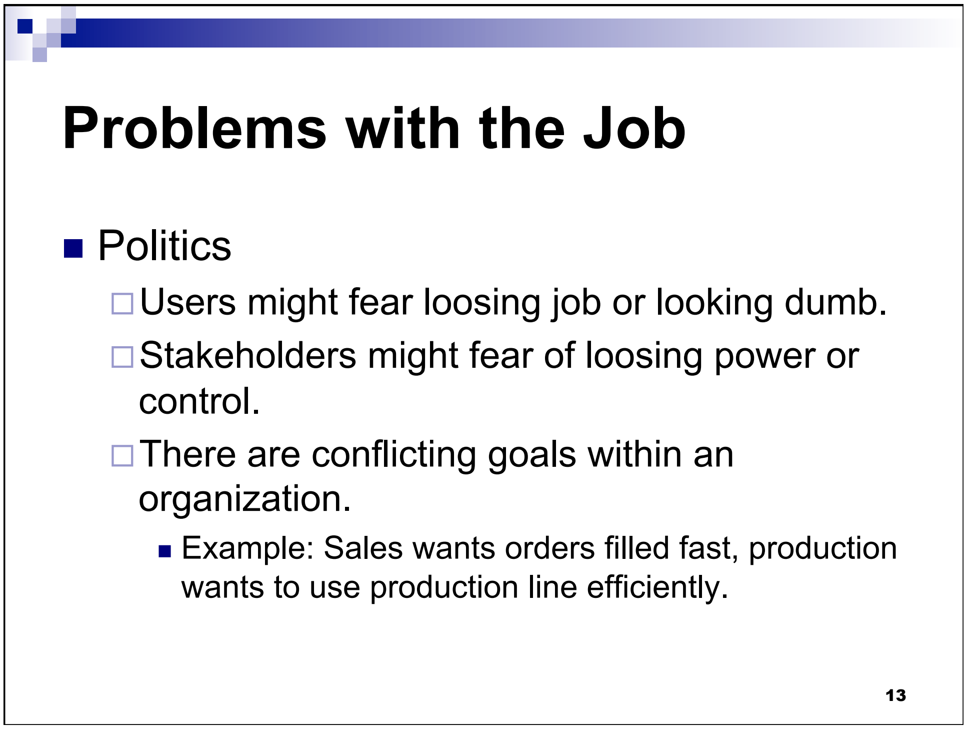
Problems with the Job

- Must make a match between what is possible and what is worth doing.
  - □ Cost/benefit decision.
- No one "right" or "wrong" solution.
  - □ Some better or worse than others; highly subjective.
  - □ Warning: this will irritate many of you.



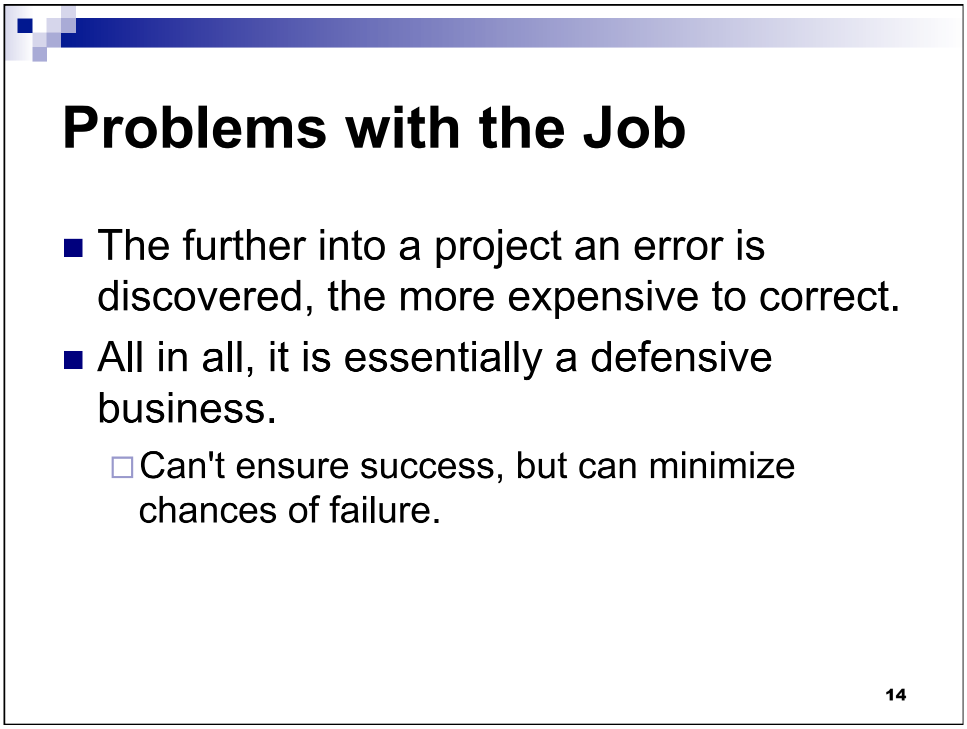
Problems with the Job

- The changing nature of computer technology.
- The changing nature of business systems.
  - □ Cannot "freeze" the specifications or the system.



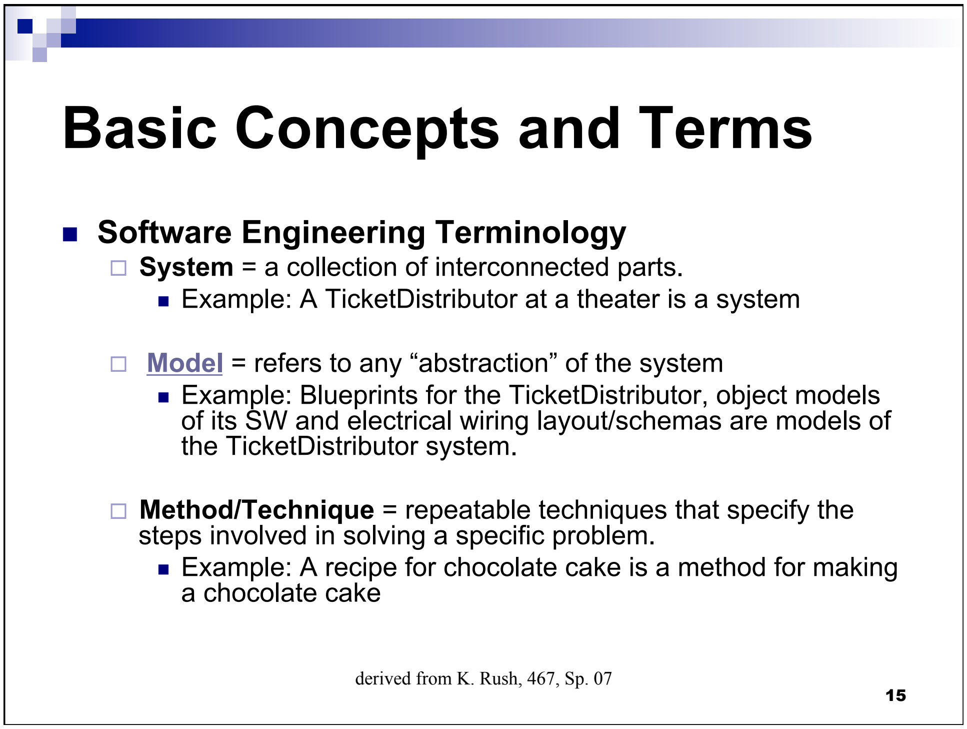
Problems with the Job

- Politics
  - Users might fear loosing job or looking dumb.
  - Stakeholders might fear of loosing power or control.
  - There are conflicting goals within an organization.
    - Example: Sales wants orders filled fast, production wants to use production line efficiently.



Problems with the Job

- The further into a project an error is discovered, the more expensive to correct.
- All in all, it is essentially a defensive business.
  - Can't ensure success, but can minimize chances of failure.

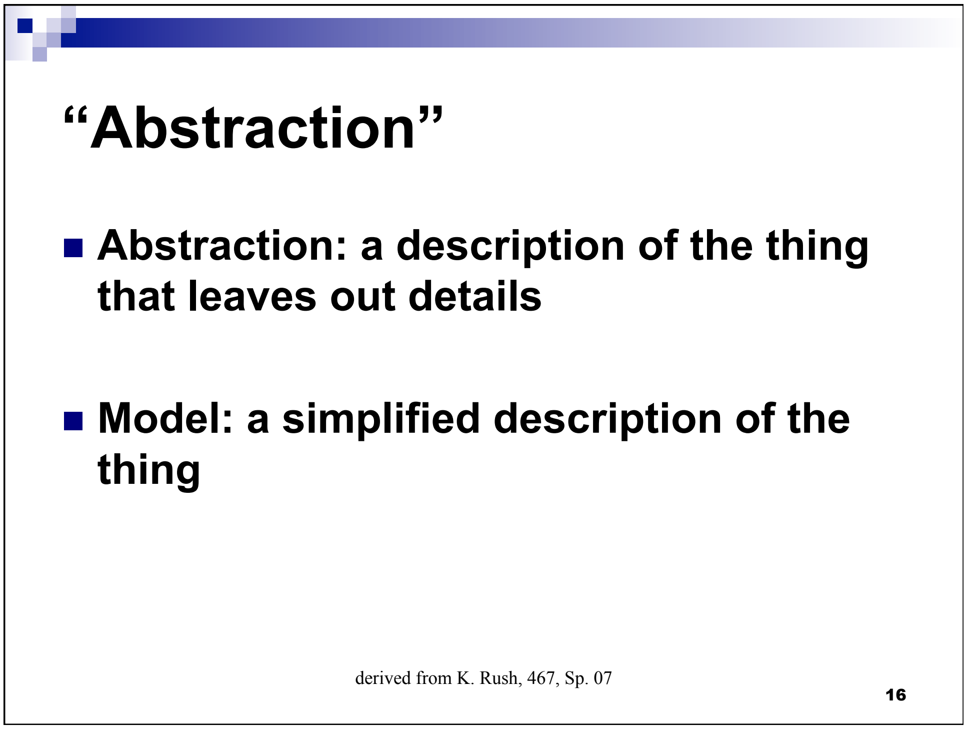


Basic Concepts and Terms

- **Software Engineering Terminology**
  - □ **System** = a collection of interconnected parts.
    - Example: A TicketDistributor at a theater is a system
  - □ Model = refers to any “abstraction” of the system
    - Example: Blueprints for the TicketDistributor, object models of its SW and electrical wiring layout/schemas are models of the TicketDistributor system.
  - □ **Method/Technique** = repeatable techniques that specify the steps involved in solving a specific problem.
    - Example: A recipe for chocolate cake is a method for making a chocolate cake

derived from K. Rush, 467, Sp. 07

15



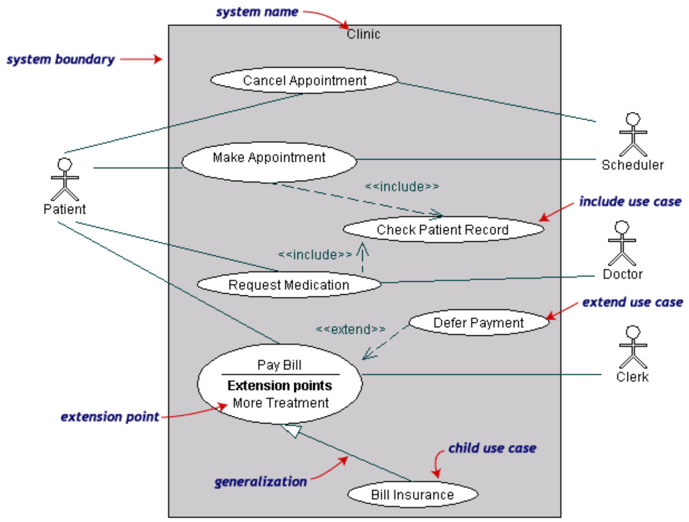
“Abstraction”

- **Abstraction: a description of the thing that leaves out details**
- **Model: a simplified description of the thing**

derived from K. Rush, 467, Sp. 07

16

Example: Model

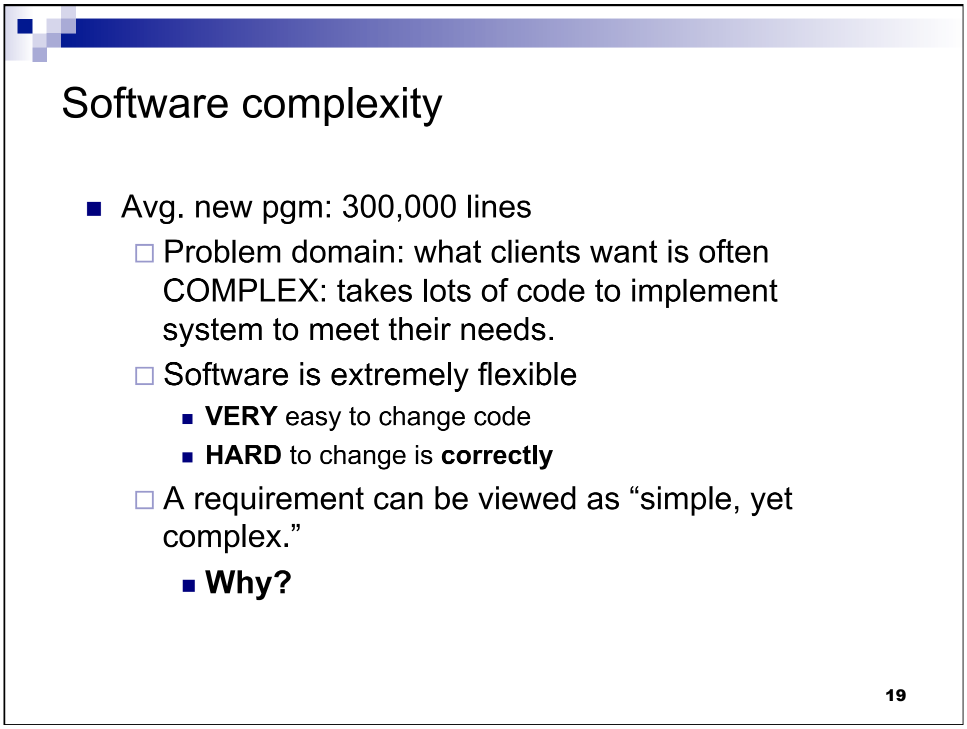


derived from K. Rush, 467, Sp. 07

Basic Concepts and Terms (Cont’d)

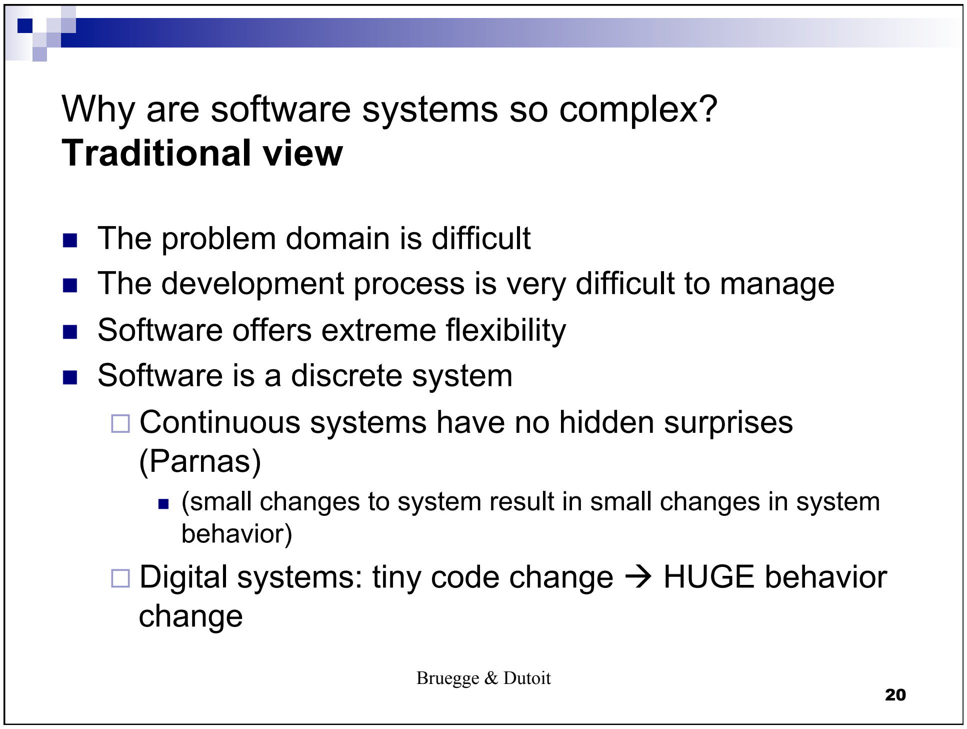
- **Functional Requirements** = all the functions the system must do (or allow a user to do)
  - □ Example: A customer shall be able to add an item to a shopping basket.
- **Non-Functional Requirements** = other characteristics or attributes the system must have
  - □ Example: The system must be available 24 - 7.
- **Problem/Application Domain** = All aspects of the user’s problem/ situation
  - □ Physical environment
  - □ The users
  - □ Work processes
- **Solution Domain** = The system: objects, processes, design, implementation

derived from K. Rush, 467, Sp. 07



Software complexity

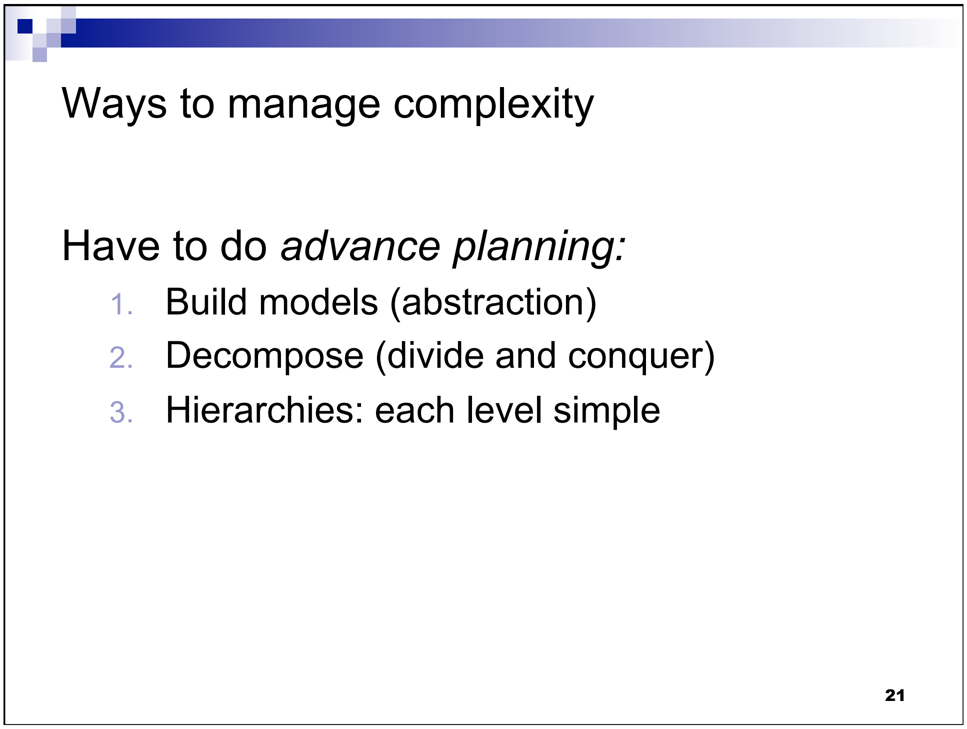
- Avg. new pgm: 300,000 lines
  - □ Problem domain: what clients want is often COMPLEX: takes lots of code to implement system to meet their needs.
  - □ Software is extremely flexible
    - **VERY** easy to change code
    - **HARD** to change is **correctly**
  - □ A requirement can be viewed as “simple, yet complex.”
    - **Why?**



Why are software systems so complex? **Traditional view**

- The problem domain is difficult
- The development process is very difficult to manage
- Software offers extreme flexibility
- Software is a discrete system
  - □ Continuous systems have no hidden surprises (Parnas)
    - (small changes to system result in small changes in system behavior)
  - □ Digital systems: tiny code change → HUGE behavior change

Bruegge & Dutoit

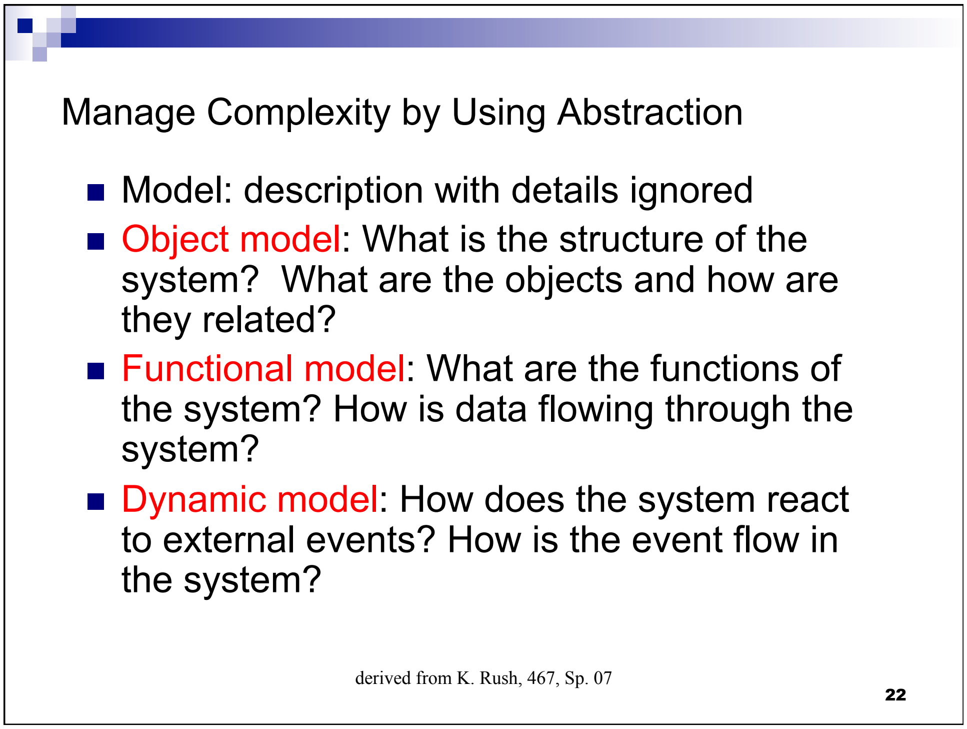


Ways to manage complexity

Have to do *advance planning:*

1. Build models (abstraction)
2. Decompose (divide and conquer)
3. Hierarchies: each level simple

21

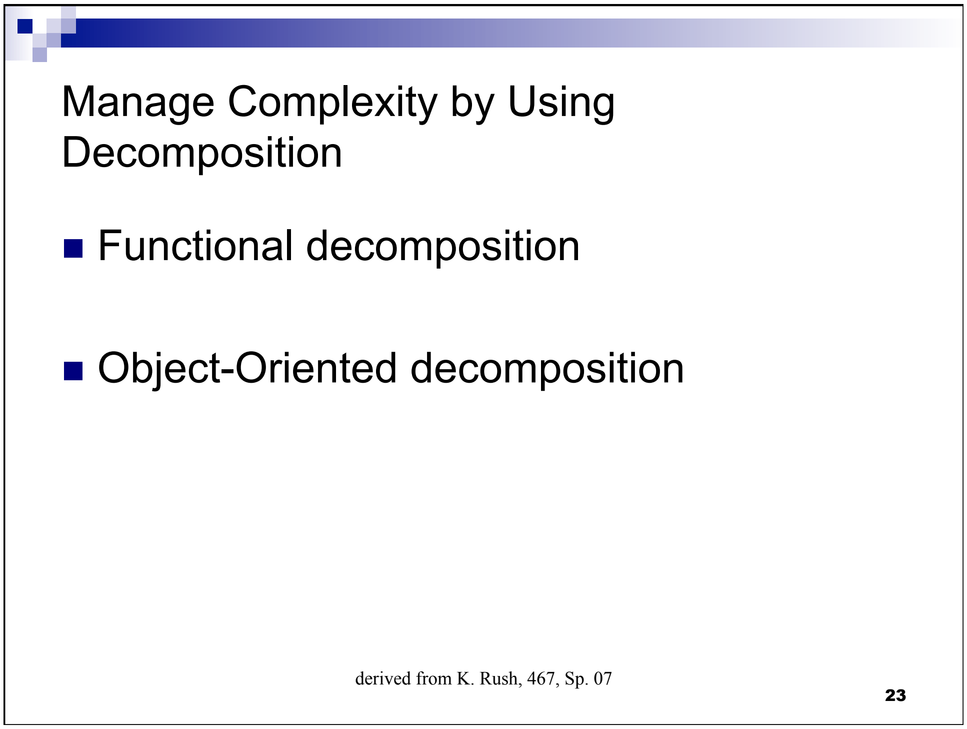


Manage Complexity by Using Abstraction

- Model: description with details ignored
- Object model: What is the structure of the system? What are the objects and how are they related?
- Functional model: What are the functions of the system? How is data flowing through the system?
- Dynamic model: How does the system react to external events? How is the event flow in the system?

derived from K. Rush, 467, Sp. 07

22

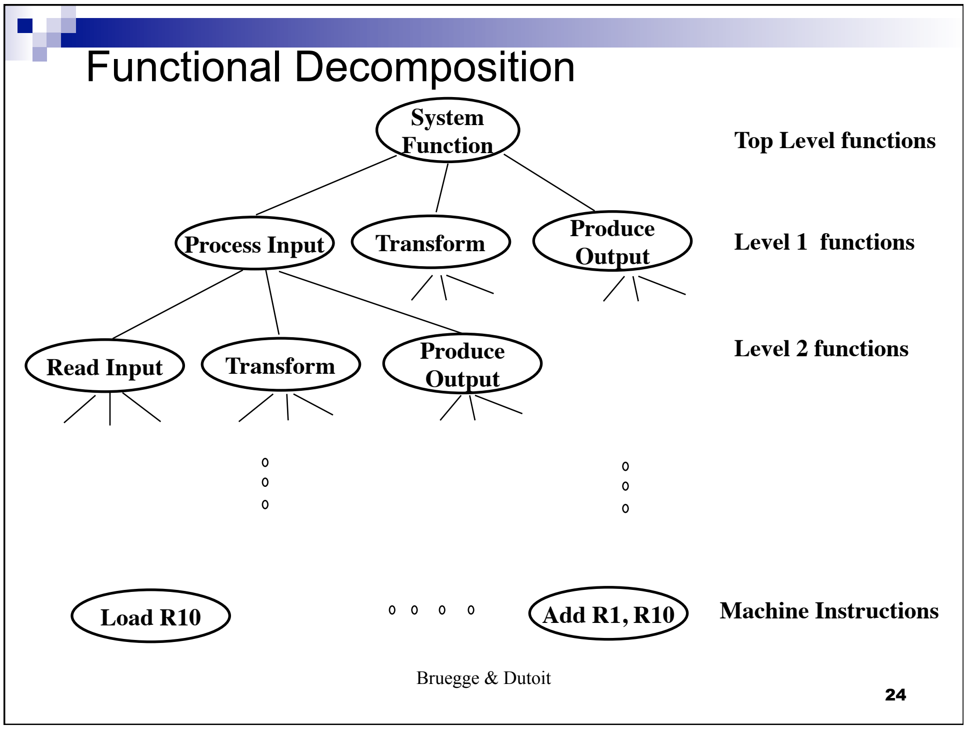


Manage Complexity by Using Decomposition

- Functional decomposition
- Object-Oriented decomposition

derived from K. Rush, 467, Sp. 07

23



Functional Decomposition

System Function

Process Input

Transform

Produce Output

Read Input

Transform

Produce Output

Load R10

Add R1, R10

Top Level functions

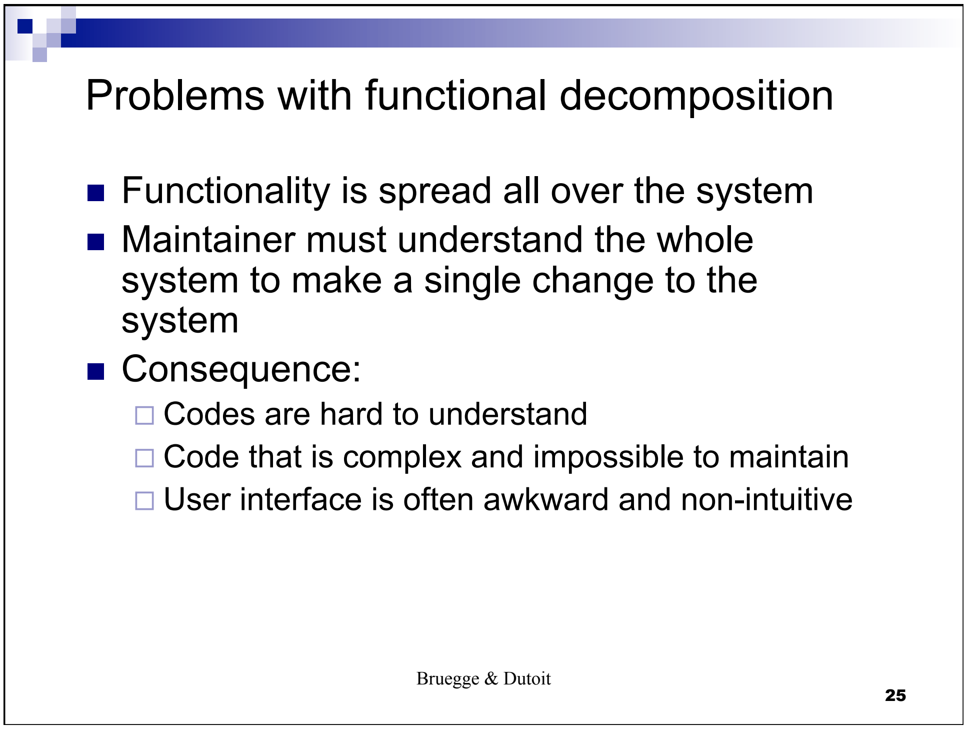
Level 1 functions

Level 2 functions

Machine Instructions

Bruegge & Dutoit

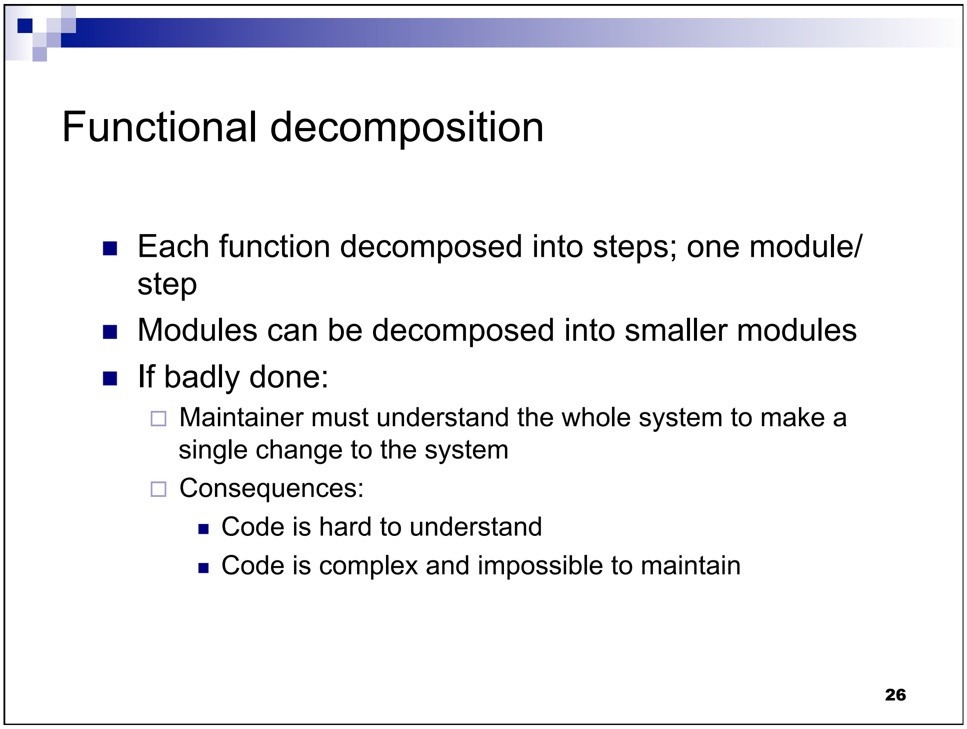
24



Problems with functional decomposition

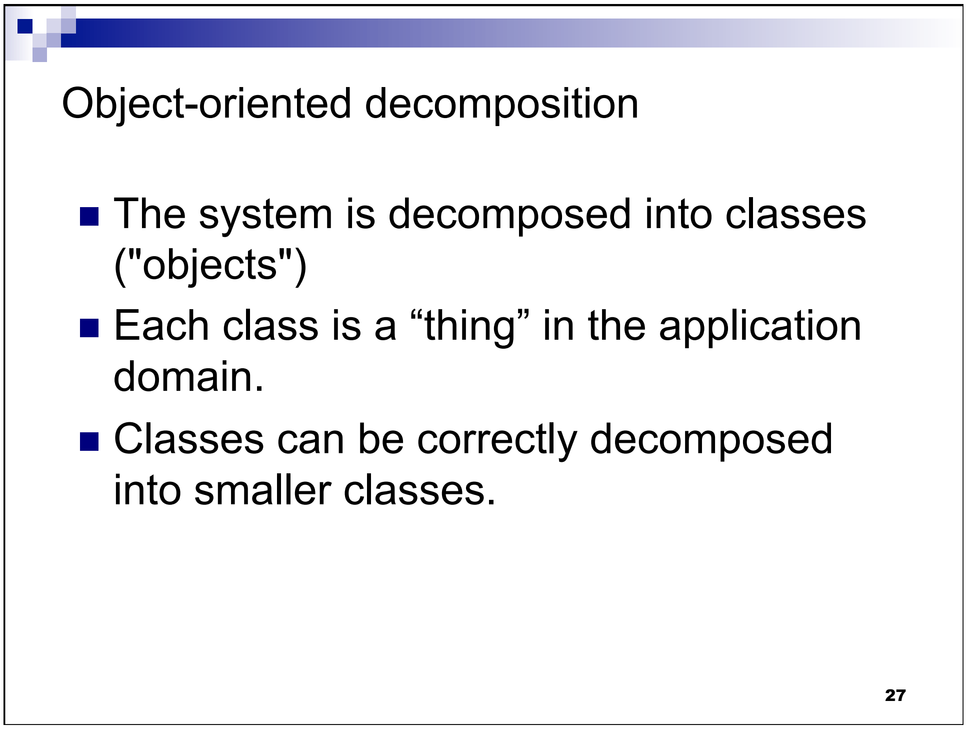
- Functionality is spread all over the system
- Maintainer must understand the whole system to make a single change to the system
- Consequence:
  - □ Codes are hard to understand
  - □ Code that is complex and impossible to maintain
  - □ User interface is often awkward and non-intuitive

Bruegge & Dutoit



Functional decomposition

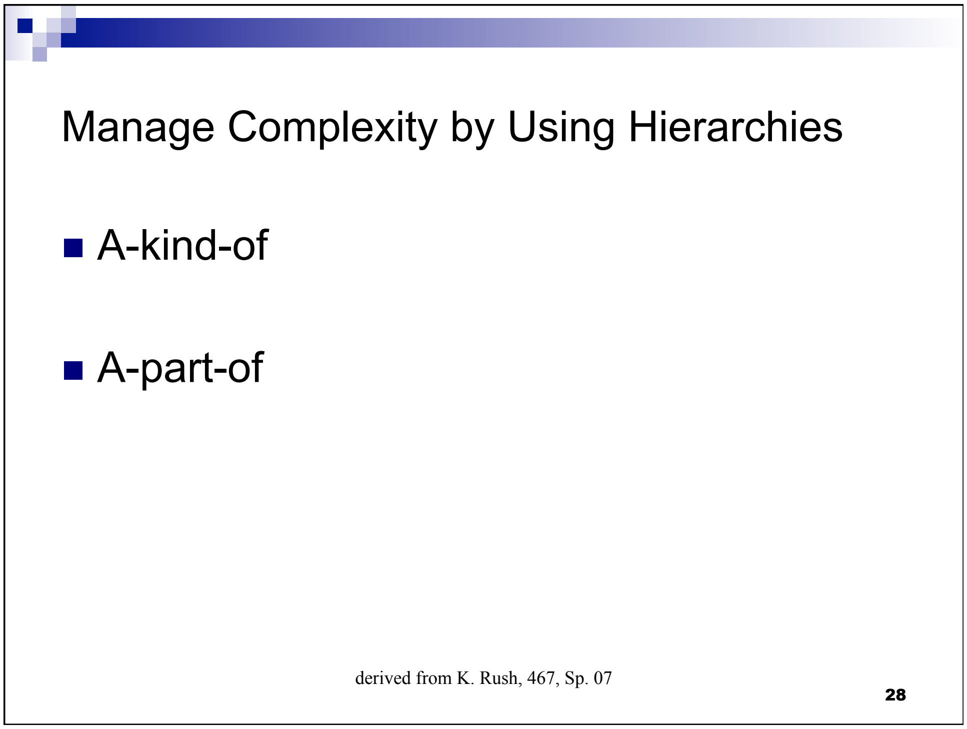
- Each function decomposed into steps; one module/ step
- Modules can be decomposed into smaller modules
- If badly done:
  - □ Maintainer must understand the whole system to make a single change to the system
  - □ Consequences:
    - Code is hard to understand
    - Code is complex and impossible to maintain



Object-oriented decomposition

- The system is decomposed into classes ("objects")
- Each class is a “thing” in the application domain.
- Classes can be correctly decomposed into smaller classes.

27

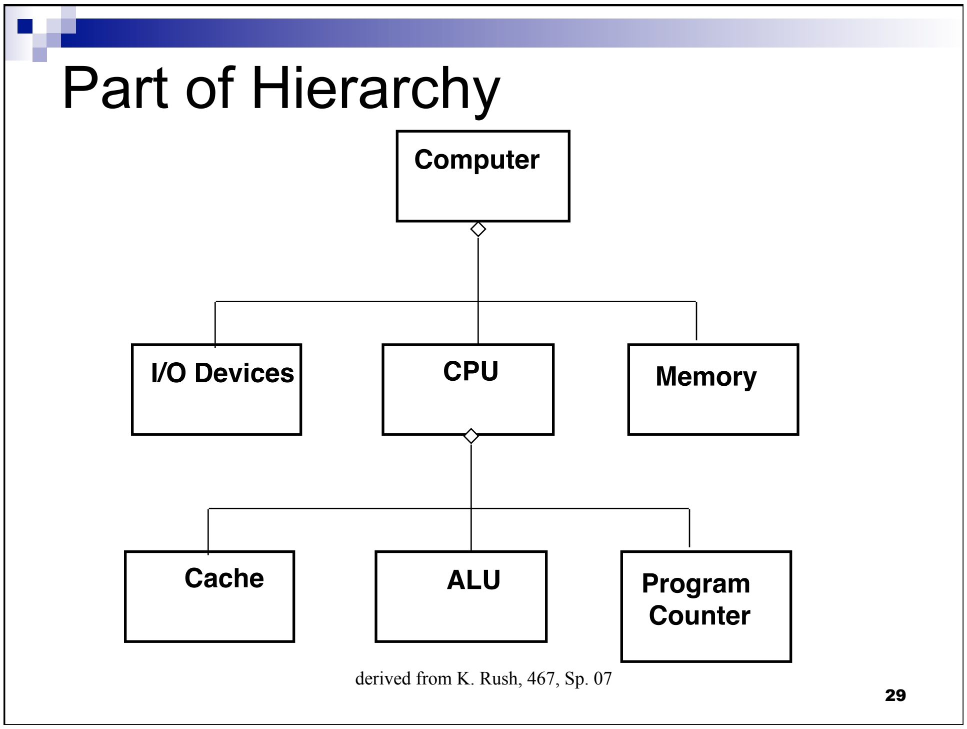


Manage Complexity by Using Hierarchies

- A-kind-of
- A-part-of

derived from K. Rush, 467, Sp. 07

28



Part of Hierarchy

Computer

I/O Devices

CPU

Memory

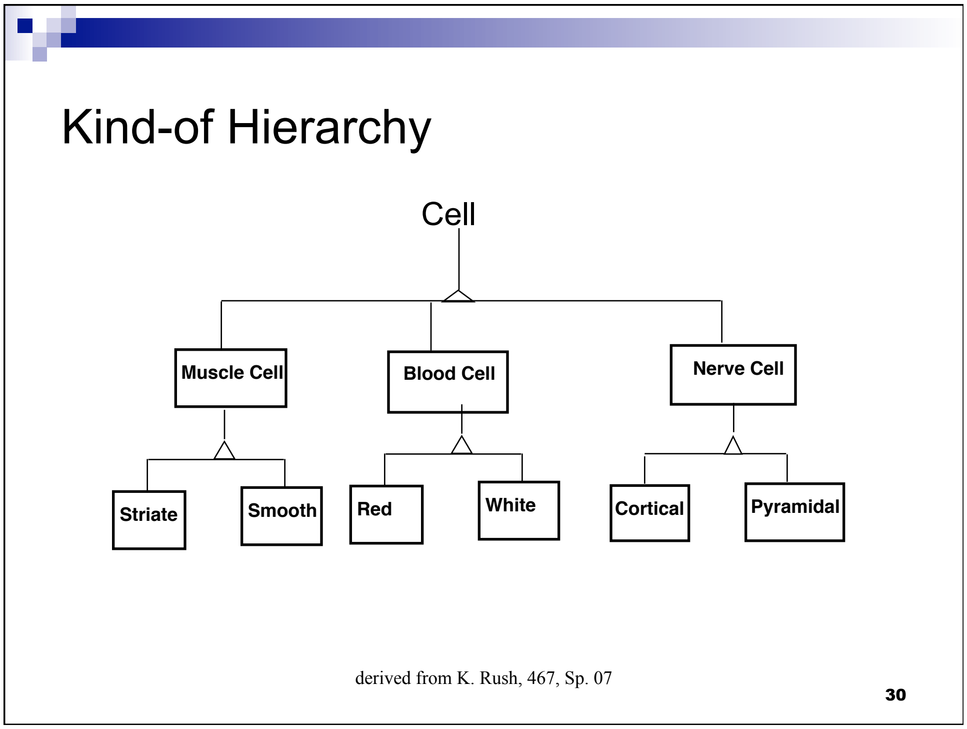
Cache

ALU

Program Counter

derived from K. Rush, 467, Sp. 07

29



Kind-of Hierarchy

Cell

Muscle Cell

Blood Cell

Nerve Cell

Striate

Smooth

Red

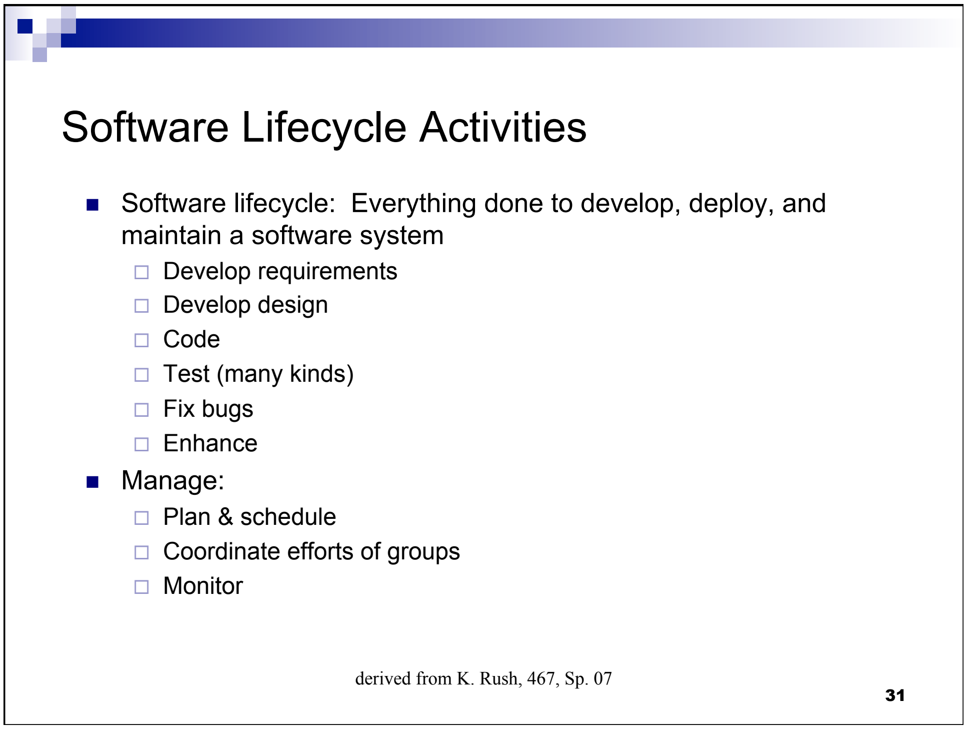
White

Cortical

Pyramidal

derived from K. Rush, 467, Sp. 07

30

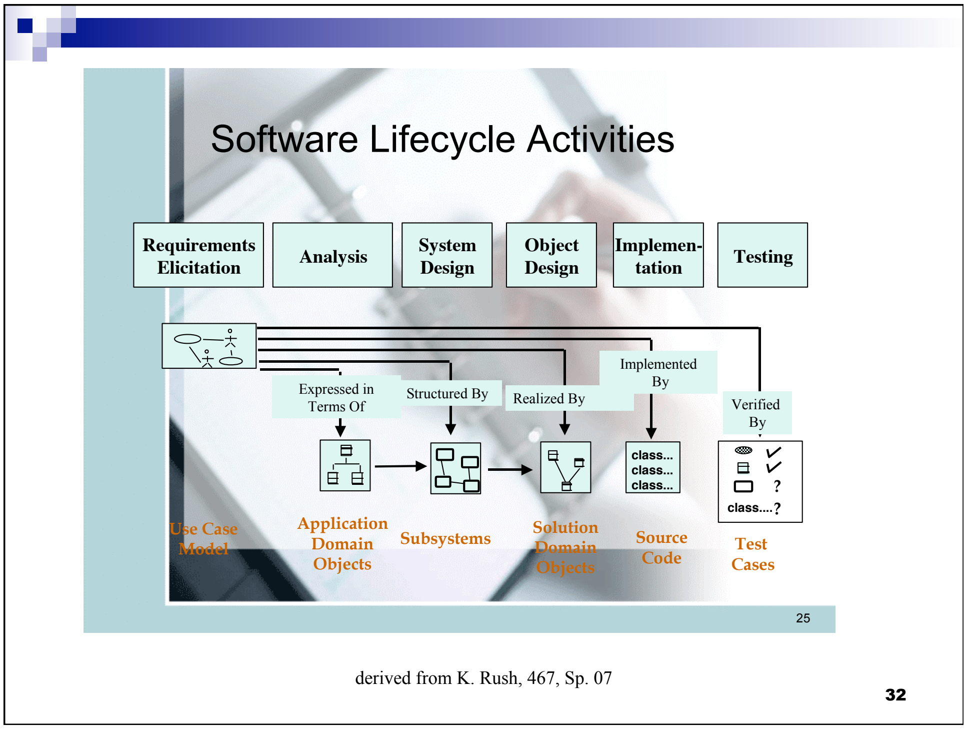


Software Lifecycle Activities

- Software lifecycle: Everything done to develop, deploy, and maintain a software system
  - □ Develop requirements
  - □ Develop design
  - □ Code
  - □ Test (many kinds)
  - □ Fix bugs
  - □ Enhance
- Manage:
  - □ Plan & schedule
  - □ Coordinate efforts of groups
  - □ Monitor

derived from K. Rush, 467, Sp. 07

31

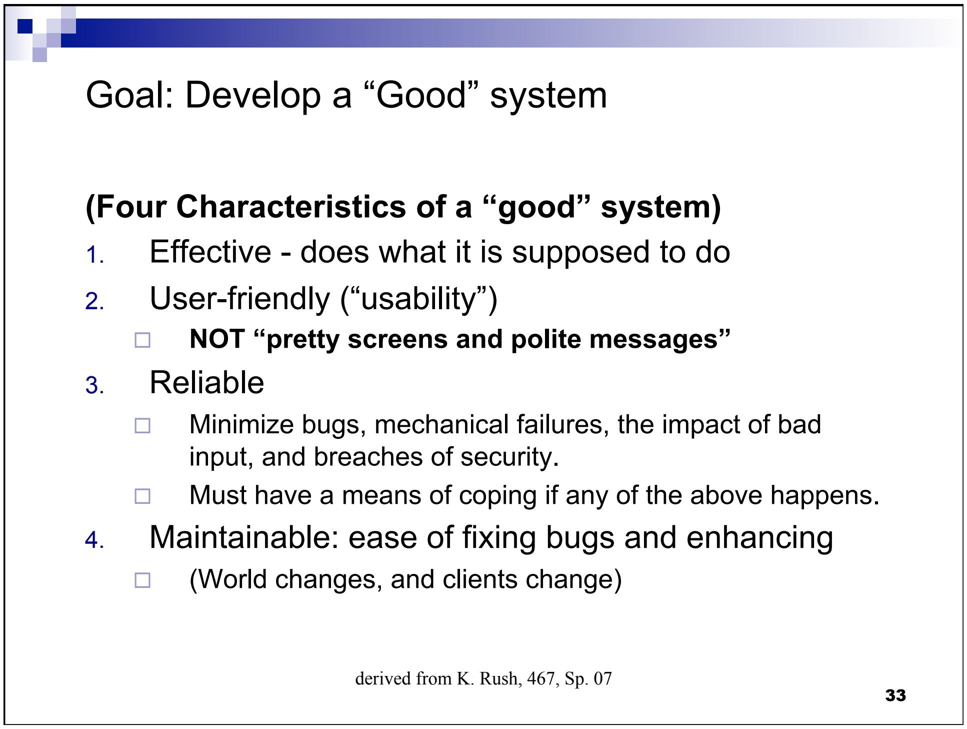


Software Lifecycle Activities

25

derived from K. Rush, 467, Sp. 07

32



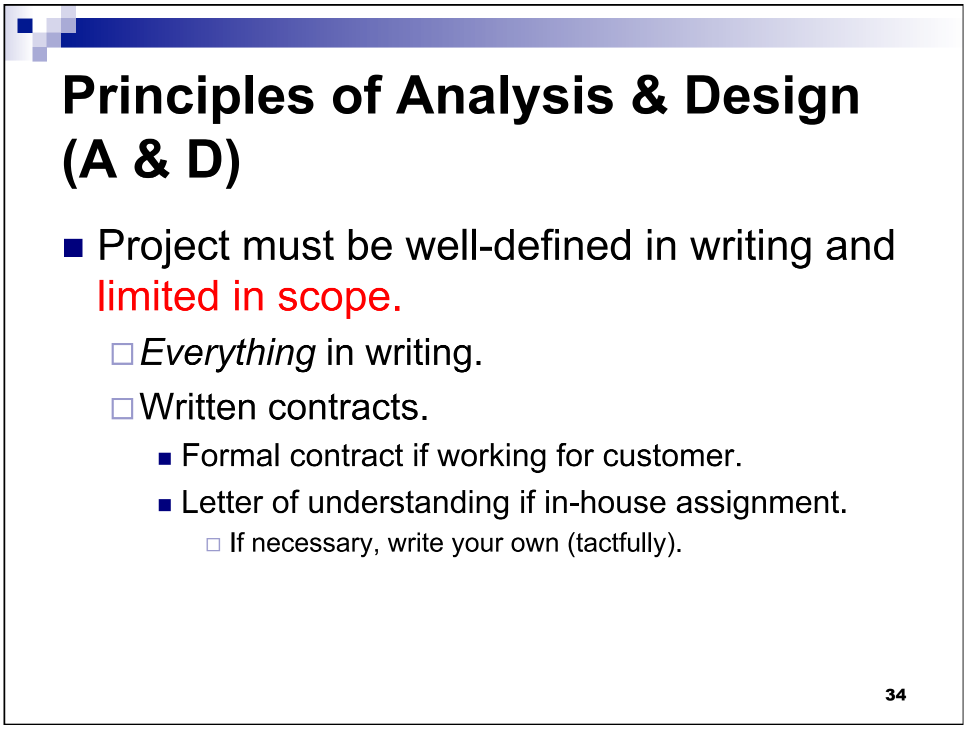
Goal: Develop a “Good” system

(Four Characteristics of a “good” system)

1. Effective - does what it is supposed to do
2. User-friendly (“usability”)
   - □ **NOT “pretty screens and polite messages”**
3. Reliable
   - □ Minimize bugs, mechanical failures, the impact of bad input, and breaches of security.
   - □ Must have a means of coping if any of the above happens.
4. Maintainable: ease of fixing bugs and enhancing
   - □ (World changes, and clients change)

derived from K. Rush, 467, Sp. 07

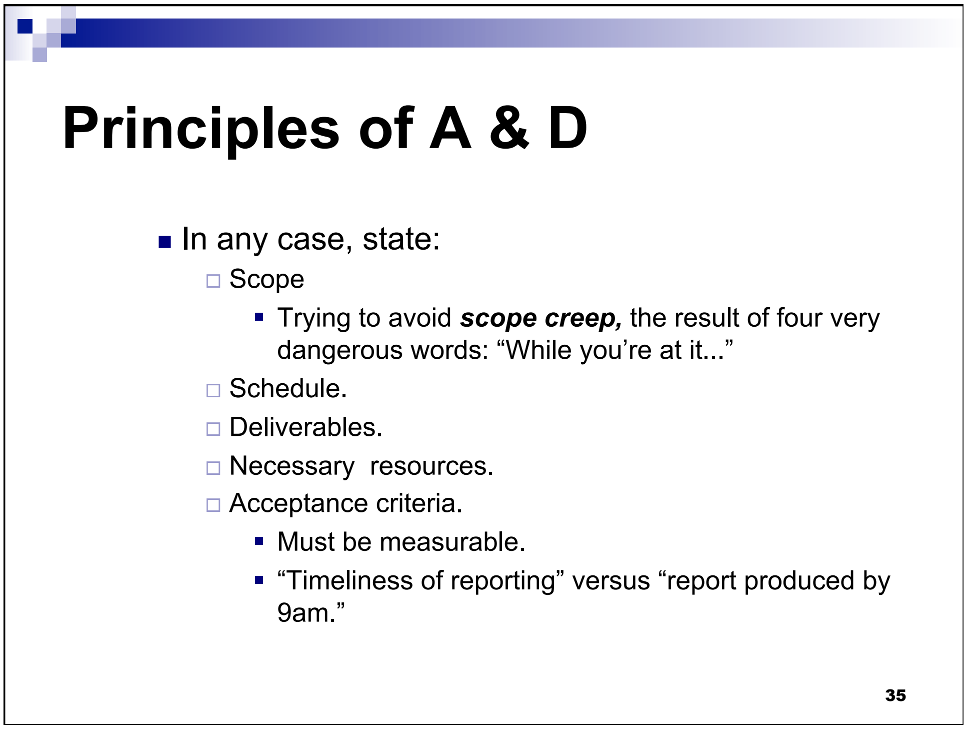
33



Principles of Analysis & Design (A & D)

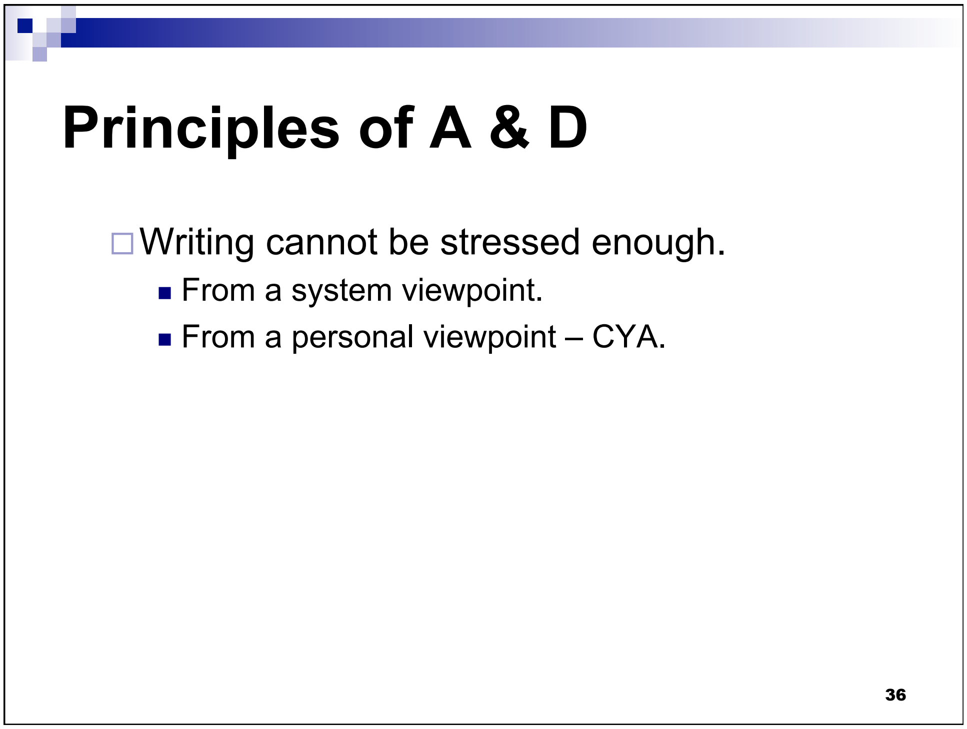
- ■ Project must be well-defined in writing and limited in scope.
  - □ *Everything* in writing.
  - □ Written contracts.
    - ■ Formal contract if working for customer.
    - ■ Letter of understanding if in-house assignment.
      - □ If necessary, write your own (tactfully).

34



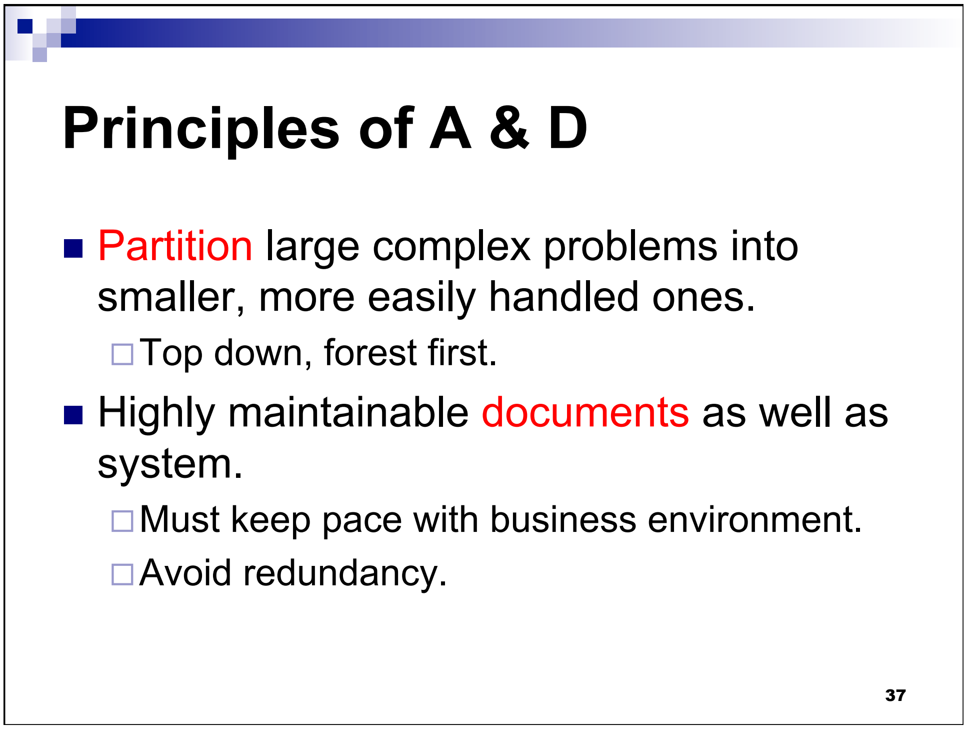
Principles of A & D

- In any case, state:
  - Scope
    - Trying to avoid ***scope creep,*** the result of four very dangerous words: “While you’re at it...”
  - Schedule.
  - Deliverables.
  - Necessary resources.
  - Acceptance criteria.
    - Must be measurable.
    - “Timeliness of reporting” versus “report produced by 9am.”



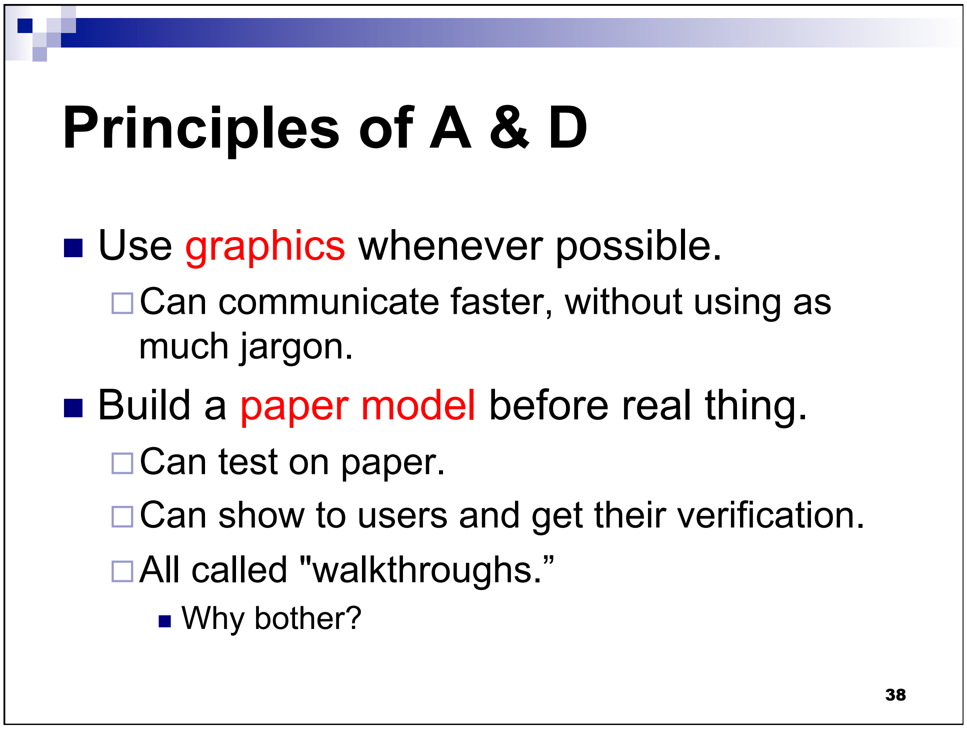
Principles of A & D

- Writing cannot be stressed enough.
  - From a system viewpoint.
  - From a personal viewpoint – CYA.



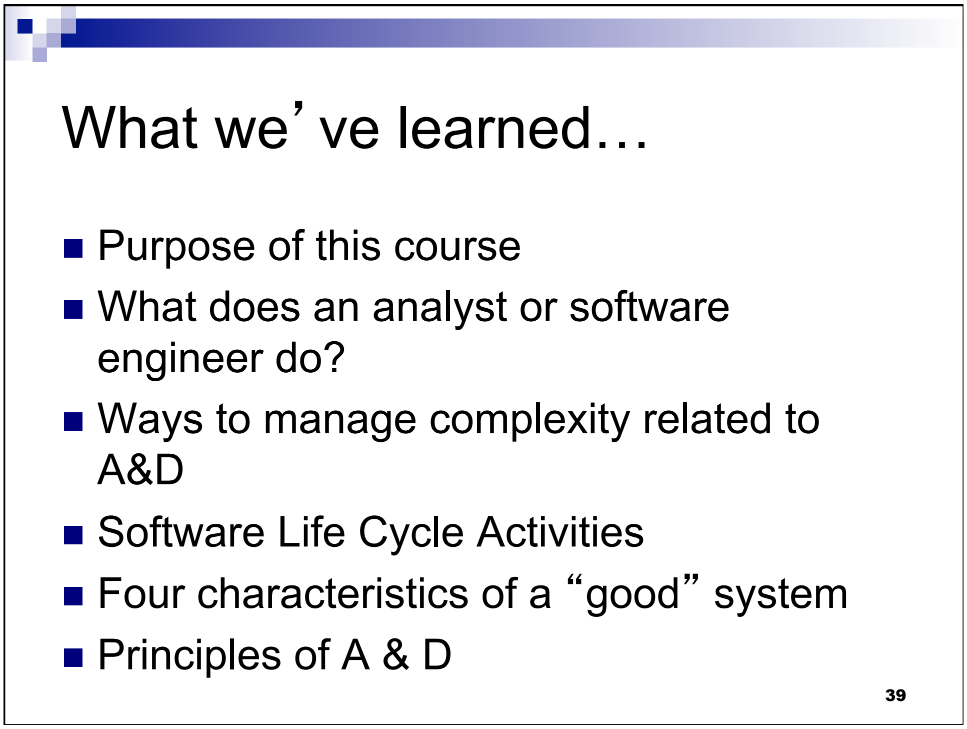
Principles of A & D

- Partition large complex problems into smaller, more easily handled ones.
  - Top down, forest first.
- Highly maintainable documents as well as system.
  - Must keep pace with business environment.
  - Avoid redundancy.



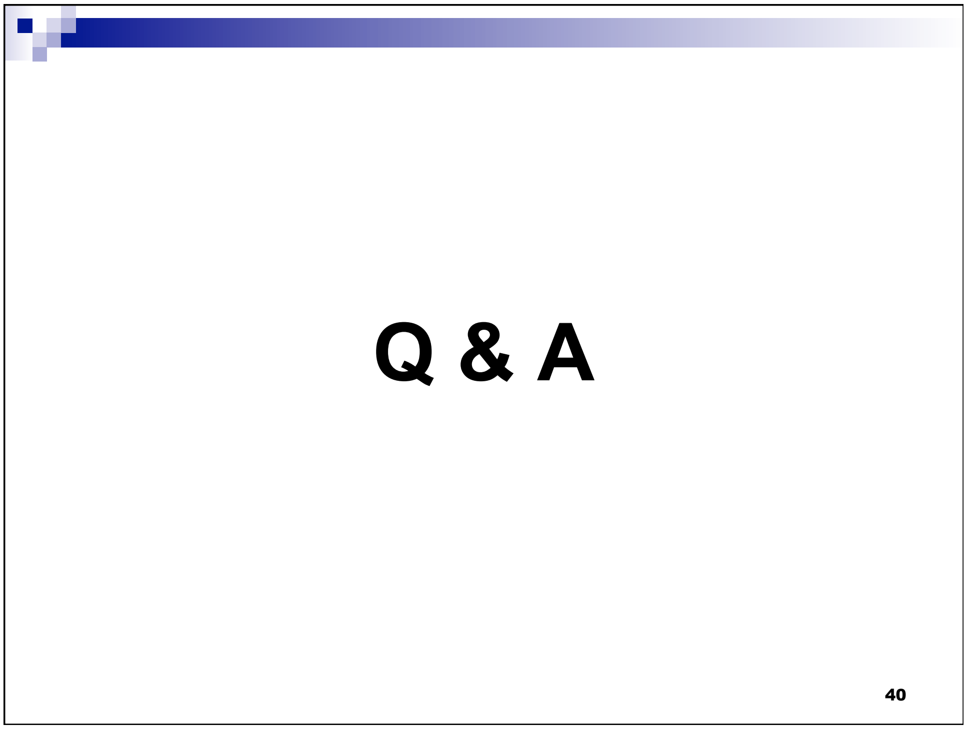
Principles of A & D

- Use graphics whenever possible.
  - Can communicate faster, without using as much jargon.
- Build a paper model before real thing.
  - Can test on paper.
  - Can show to users and get their verification.
  - All called "walkthroughs.”
    - Why bother?



What we’ve learned…

- Purpose of this course
- What does an analyst or software engineer do?
- Ways to manage complexity related to A&D
- Software Life Cycle Activities
- Four characteristics of a “good” system
- Principles of A & D



Q & A