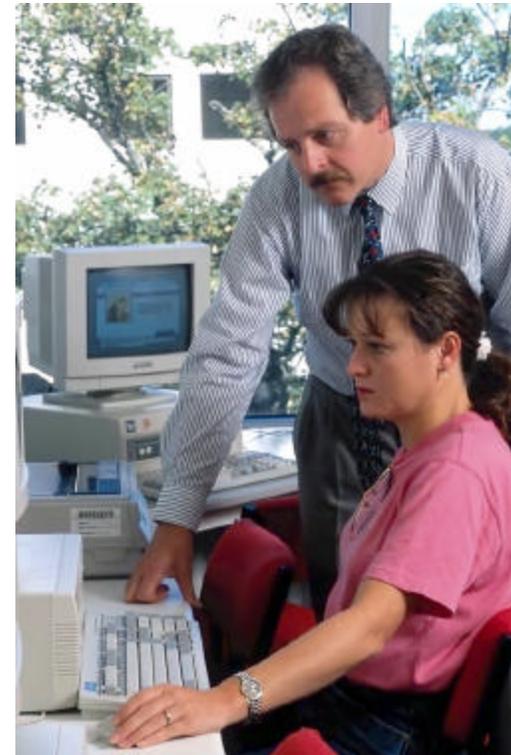


4. Information Systems

UNDERSTANDING ITS/CVO TECHNOLOGY APPLICATIONS

Student Manual

MODULE 4 - INFORMATION SYSTEMS



US Dept of Transportation

Module 4 - Information Systems

Title

Learning Objectives

You will be able to:

- Explain the components and architecture of an information system and how data flows in that system
- Understand the information systems development process and some related issues

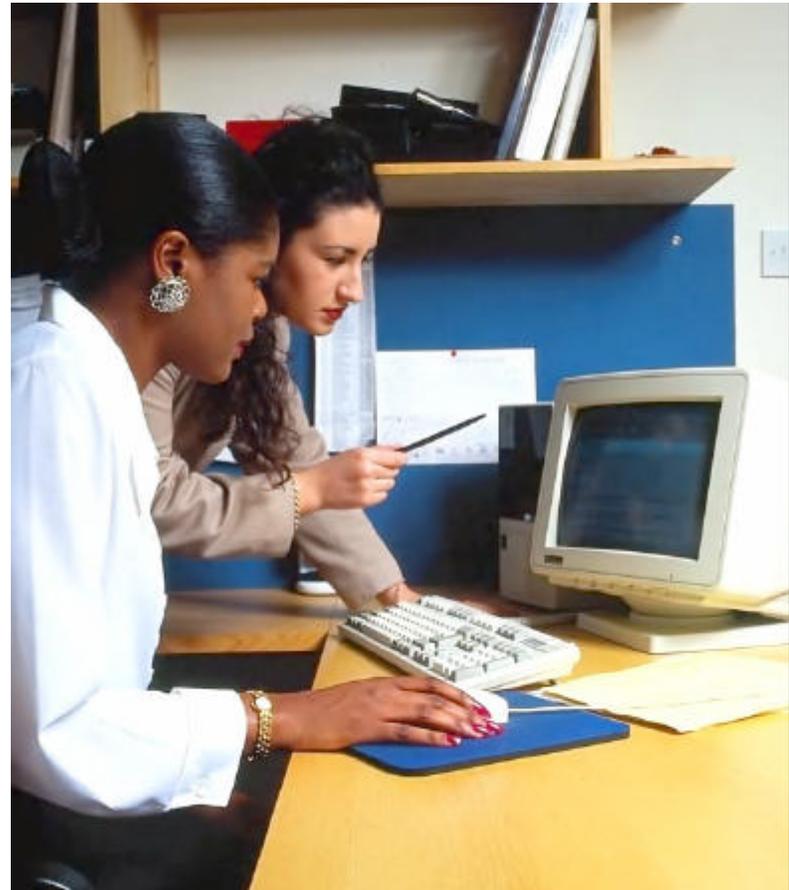
Module Structure

- What is an information system?
 - Information Systems
 - Computer Systems
 - A Simple Information System Model
- System Development Example: *CVO FORMS*
 - User Interfaces
 - Data Definitions & Databases
 - System Interfaces
- System Demonstration: *CAT*
- System Development Issues
 - Build vs. Buy
 - Life Cycle Models
 - Integration & Test
 - Configuration Management
- Discussion: *How has your state developed or acquired information systems in the past?*
- Questions & Wrap-up

A good basic reference on information systems:
Introduction to Information Systems, James A. Obrien, Irwin/McGraw-Hill, 1997.

Information systems are more than just computers!

An information system is an organized combination of people, hardware, software (including procedures & other documentation), communications networks, and data resources that collects, transforms and disseminates information to meet some need.



Information systems are used in many ways to support business operations

- Information systems may support
 - Operations
 - Management
- What systems can you think of that fit into those broad categories?

Information systems involve dozens of specialized technologies



Think about technologies apparent to the user and those invisible to the user. *Did we capture all of them in the previous brainstorm on technologies?*

What Technologies are Involved in Information Systems?

- **Topics Mentioned in This Module**

- Hardware
- Personal Computers
- Servers
- Mainframes
- Software
- Operating Systems
- Graphical User Interface (GUI)
- Databases
- Data Base Management Systems (DBMS)
- Software Development Process
- Legacy Systems
- Commercial Of-the-Shelf (COTS) Software
- Package vs. Custom Life Cycle

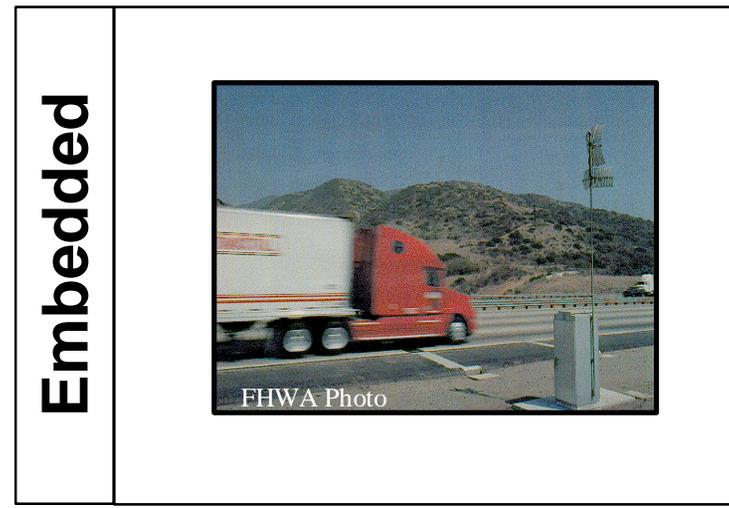
- **Topics Covered in the Next Module**

- Local Area Networks (LANs)
- Wide Area Networks (WANs)
- Wireless Communications
- Electronic Data Interchange (EDI)
- Client / Server
- Internet Communications
- World Wide Web

- **Topics Not Covered in This Course**

- Hardware Design
- Software Engineering Management
- Programming Languages
- Compiler Design
- Software Metrics
- Numerical Methods
- Artificial Intelligence
- ...
- **& 100's of other specialties!**

Computer systems are an interconnected set of hardware elements that can execute software programs



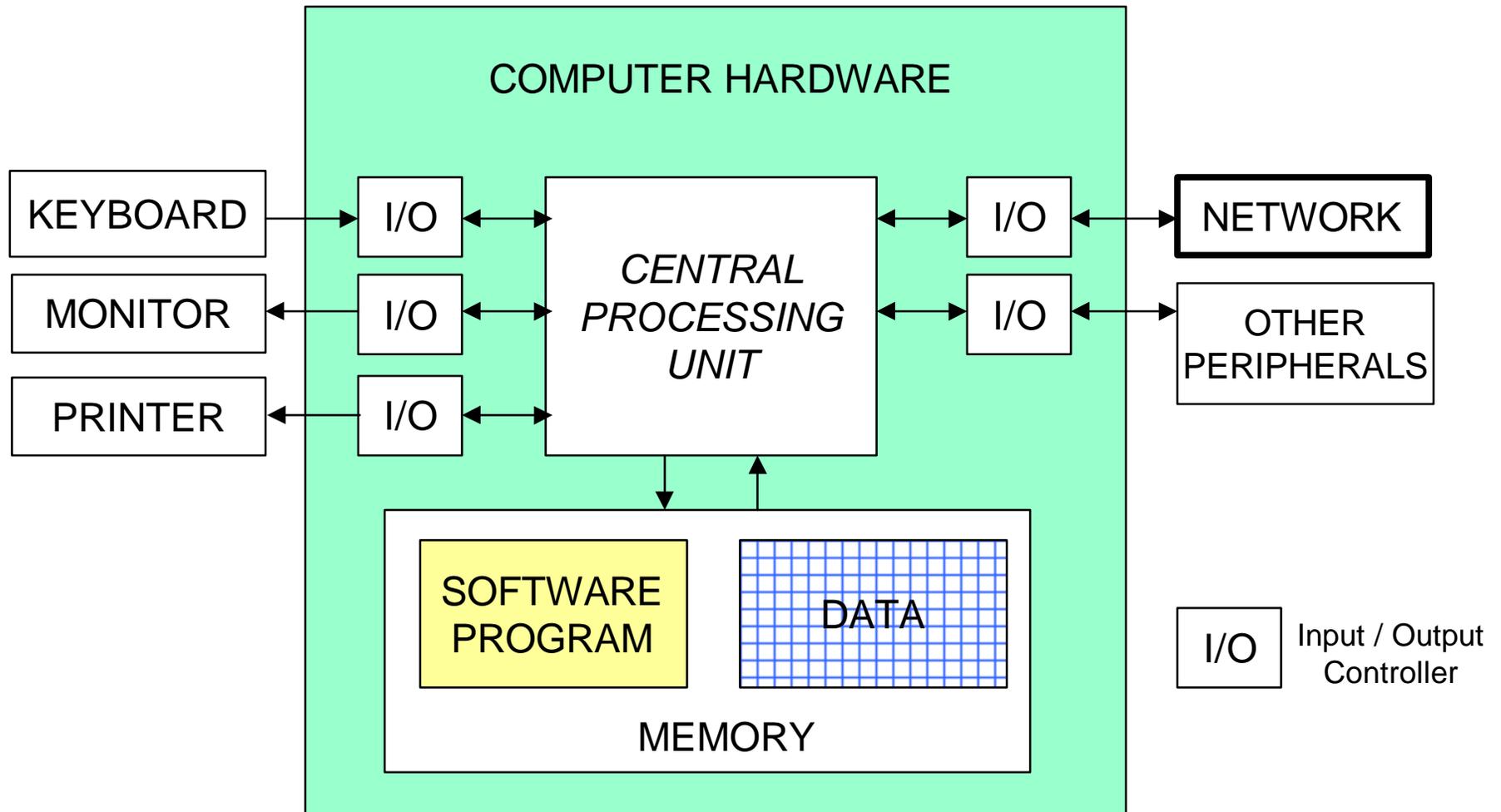
Computer systems vary greatly in size, complexity and cost

	Workstation	Server	Mainframe
Size	Desktop	A small cabinet.	Several Large Cabinets
Cost	\$1-5K	\$5-50K	\$100K- 5M
System Speed	100 - 400 MIPS	100 - 1600 MIPS	1000 - 40000 MIPS
Processors	1	1-4	10's of specialized processors
Memory	32-64M	256-512M	1-2G
Disk Storage	1-8G	10-50G	100'sG
Input/Output Channels	3-8, >.01-3Mhz	3-8+, >1-100Mhz	10's, > 1GHz each
Simultaneous Users	1	10's	100's
Example Applications	Desktop, portable, word processing & other office applications, e-mail, internet access, server & mainframe access, ...	Departmental server, network server, e-mail, file sharing server, single shared application, communications processor, ...	Accounting, customer transaction processing, data warehousing, banking, finance, ...

Notes:

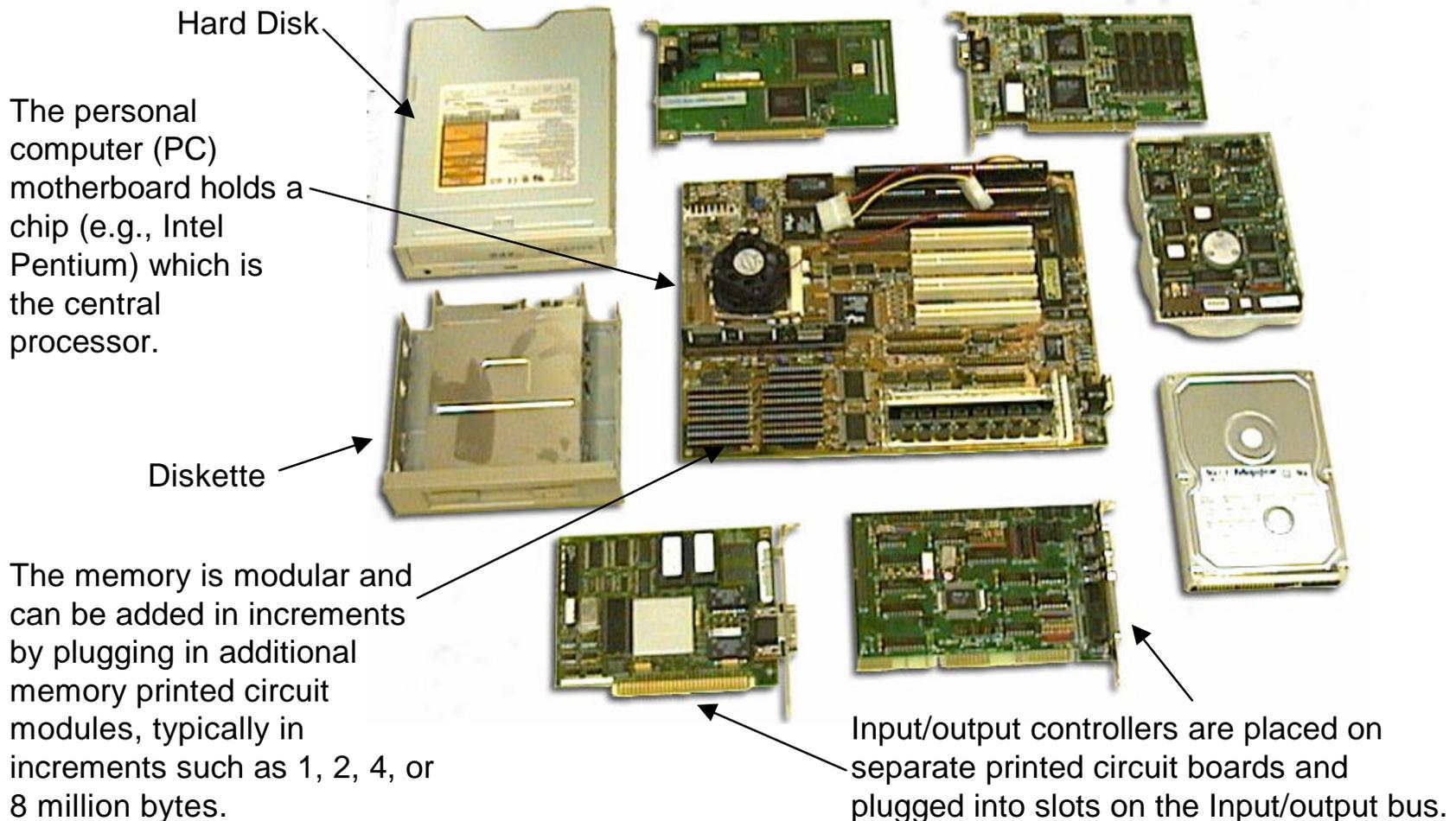
- Mainframe computers have complex architectures and many features that aren't reflected in this chart that allow them to support 100's of users on many applications simultaneously
- All values are very rough estimates of typical values
- MIPS: millions of instructions per second
- MHz: millions of Hertz, i.e., millions of bits or bytes of data per second

A Simple Computer Hardware Model

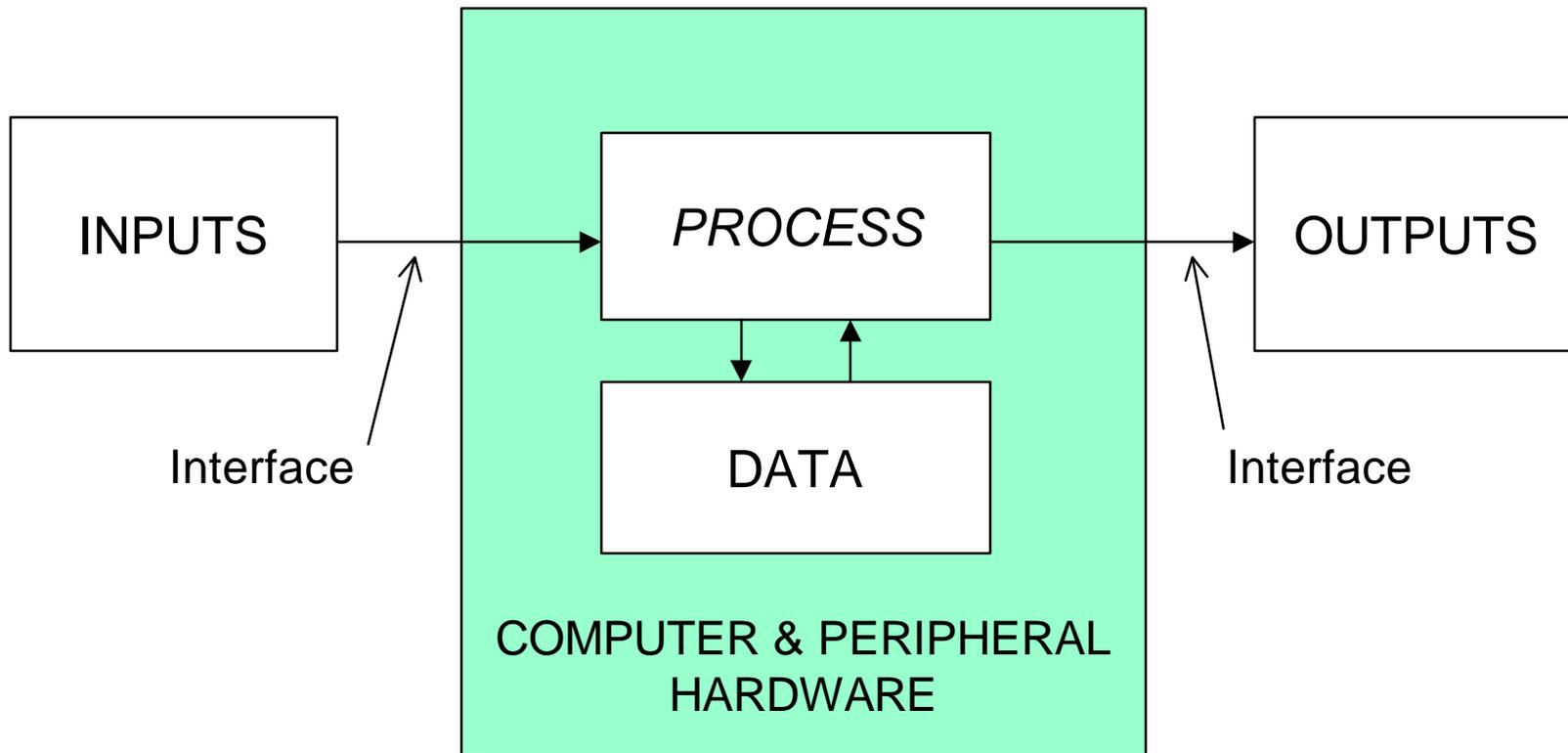


This simple model applies most directly to personal computers

This simple model can be used to think about any computer, from mainframes to personal computers (PCs). However, it actually corresponds fairly directly to the physical parts of a simple PC.



A Simple Information System Model



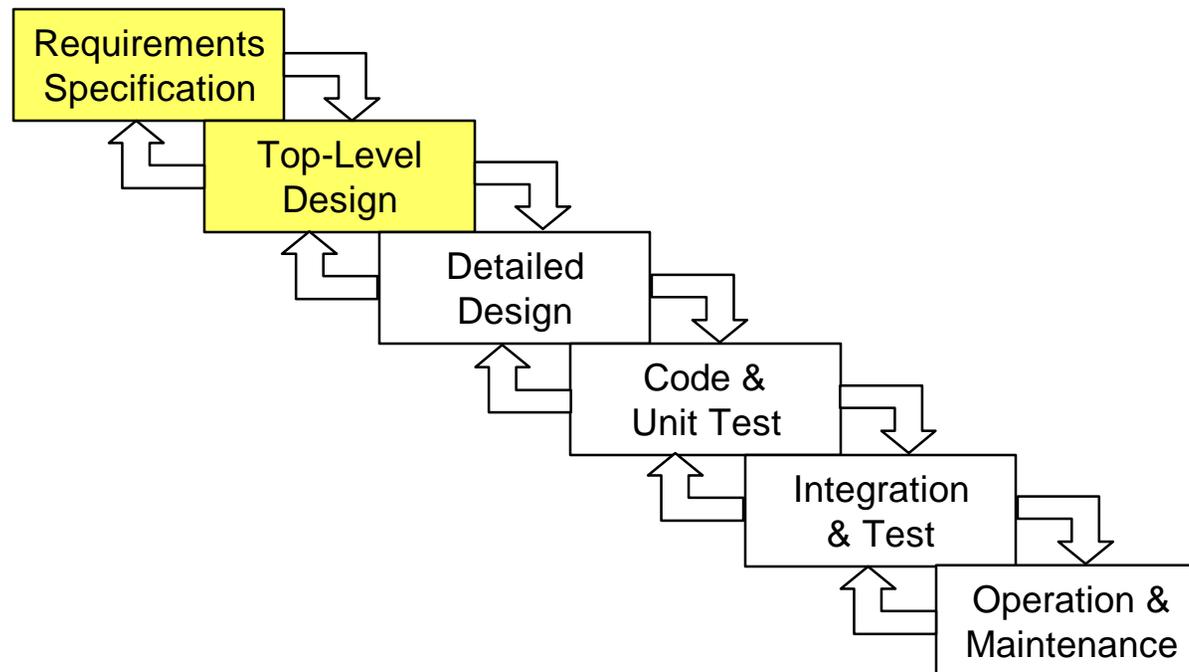
Any system can be analyzed using this simple model

SYSTEM	INPUTS	PROCESS	OUTPUTS
ASPEN	Inspection Data	Validity Checks Formatting Storage & Retrieval	Complete Inspection Report
CAT (Carrier Automated Transaction)	Credential Data	Validity Checks Formatting Storage & Retrieval EDI translation	Credential Application
CI (Credentialing Interface)	Transactions from CATs	Validity Checks EDI translation Transaction Storage	Transactions to legacy systems
SAFER (Safety & Fitness Electronic Records)	Snapshot segment transactions Subscription requests	Form snapshots Store snapshots Take & fulfill subscriptions	Snapshots

A Software Program Development Example: Filling Out *CVO Forms*

Problem: Assume you decide to build an information system to help you fill out all the forms you must deal with.

You decide to go through these steps:



Software development is often conducted in a series of steps. This provides visibility into progress.

- Requirements Specification - Determine & document what the user/customer wants the software (or system) to do.
- Top-Level Design - Determine & document the major components of the software. Determine and document the major data elements.
- Detailed Design - Determine & document the details of the software structure, including all units. Specify the step-by-step procedure to be executed by each unit.
- Code & Unit Test - Write the lines of code for each lowest level unit (i.e., component, module) of software and test it in a stand-alone manner.
- Integration & Test - Bring the software units together into one program and test progressively larger pieces.
- Operation & Maintenance - Use the software. Update it to improve it or accommodate changes in requirements or technology.

CVO FORMS: Define the Requirements

- Requirements
 - Display blank forms on the monitor
 - Accept input data from a keyboard for IRP, IFTA, titling, and intrastate registration
 - Print completed forms on a laser printer, ready for submittal to Midland
- Design Constraints
 - Run on an Intel Pentium, Windows 95 Computer System

Define functional requirements (a.k.a. business requirements) & minimize design constraints

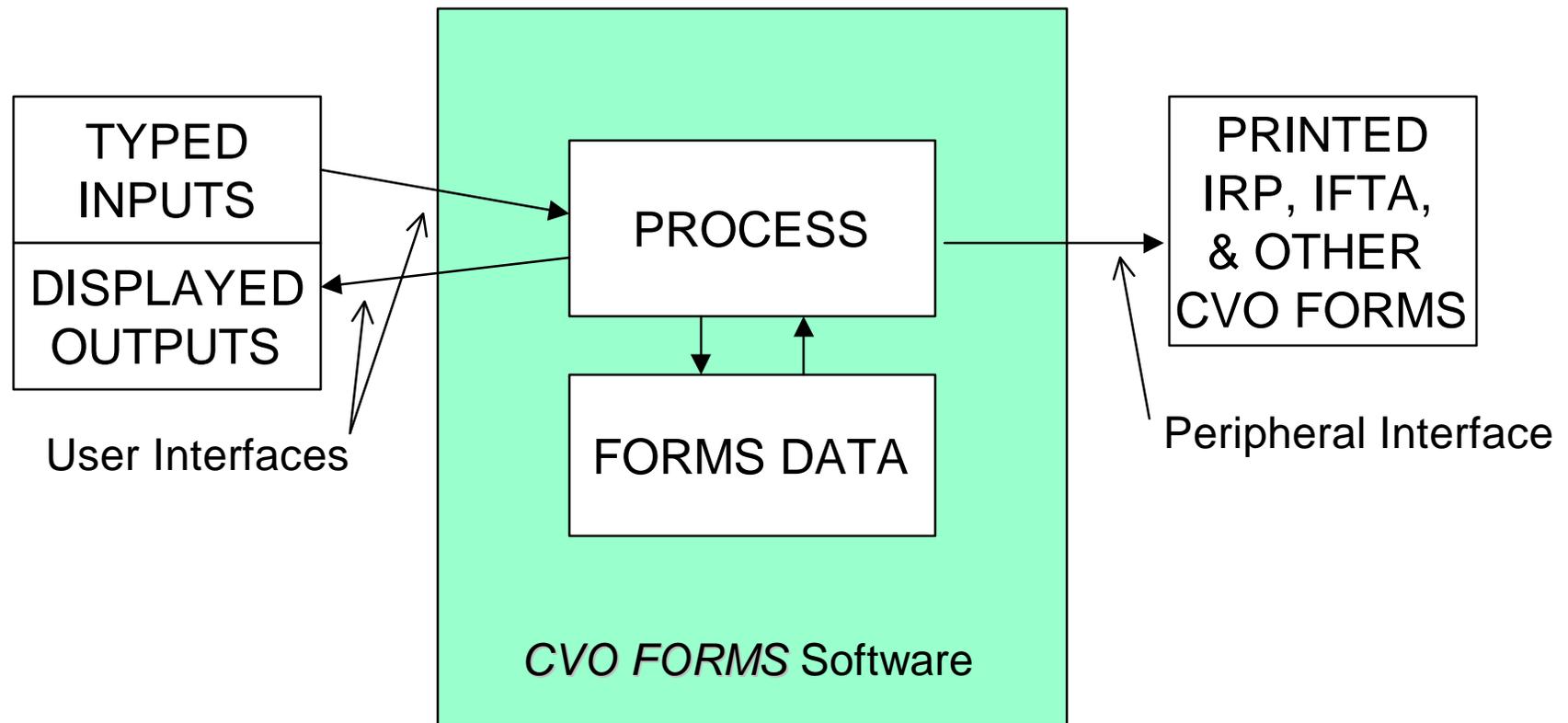
- What Constitutes “Good” Requirements Analysis (*from IEEE Std 830-1993*)

Correct Complete Consistent Unambiguous
Verifiable (Testable) Modifiable Traceable
Ranked for Importance and/or Stability

- Design Constraints
 - Include only essential design constraints
 - Typical examples
 - Physical requirements (e.g., must run on a certain machine)
 - Software development standards (e.g., must use a certain language)
 - Environment (e.g., must use a certain operating system or DBMS)
 - Including unnecessary constraints will limit the options available to the designer and may result in increased cost or lower quality

CVO FORMS Top-Level Design: Define a first cut at a Top-Level Design

A first cut at a top-level design:



The top-level design defines the major components of the software and the major data elements.

- Determine & document the major components of the software. Determine and document the major data elements.
- The top-level design of software is sometimes called the software architecture or the preliminary design.
- In each area of design, look at alternative design solutions, evaluate them against a set of criteria, and pick the best.
- Many methodologies have been developed for doing software design. Some examples:
 - Modern Structured Design
 - Object Oriented Design
 - Information Engineering
 - Prototyping
 - Rapid Application Design
 - Relational Database Design
 - ...

Reference on Design Methods:
System Analysis and Design Methods, Jeffrey L. Whitten & Lonnie Bentley, Irwin/McGraw-Hill, 1998.

CVO FORMS Top-Level Design: Define a User Interface Design

Principles of good user interface design:

- User, Product, Task, & Work
- Flow Compatibility
- Consistency
- Familiarity
- Simplicity
- Direct Manipulation
- Control
- WYSIWYG
- Flexibility
- Responsiveness
- Invisible Technology
- Robustness
- Protection
- Ease of Learning
- Ease of Use

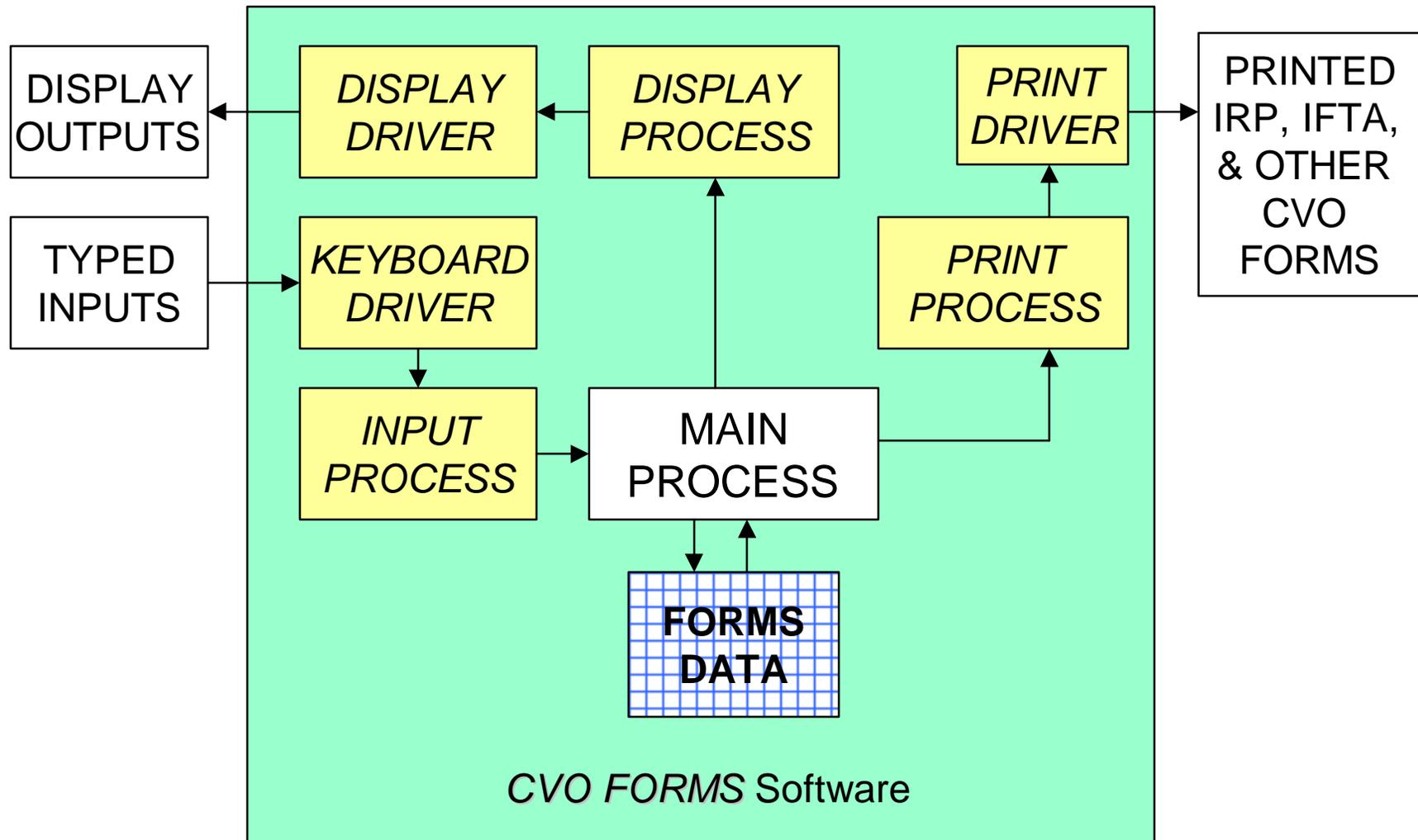
Good user interfaces require a lot of careful design work. User interface design is a specialty of its own. Principles of good user interfaces have been determined through several decades of industry experience.

Items above taken from reference below.

Principles Of Good User Interfaces

- The items on the previous page were taken from this reference: *Principles and Guidelines in Software User Interface Designs*, Deborah J. Mayhew, Prentice Hall, 1992.
- The state-of-the practice today offers many GUI (Graphical User Interface) environments, each with its own standards and guidelines. Some examples of these include:
 - Microsoft Windows
 - World Wide Web
 - X- Windows
 - Company proprietary standards
 - Product-line standards

CVO FORMS Top-Level Design: Refine the User Interface Software Top-Level Design

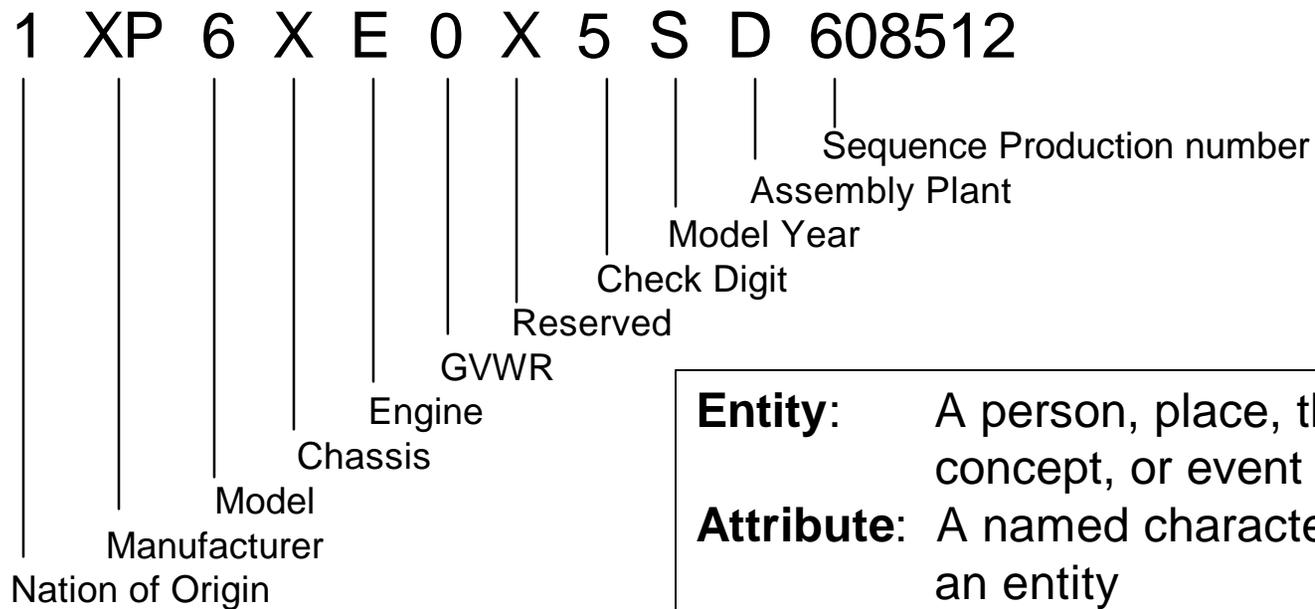


The *User Interface Software Design* is distinct from the *User-Interface Design*, although the two often proceed together

- In this step we refine the top-level software design.
- We add several commercial-off-the-shelf (COTS) software elements:
 - Display Driver, Keyboard Driver, Print Driver
- We split the process function into several modules:
 - Input Process, Main Process, Display Process, Print Process
- We determine that this refinement is an improvement over the previous alternative, because it saves time (by using existing drivers) and is more modular (usually making it more build-able, testable, & maintainable).

A critical aspect of developing information systems is to have **precise data definitions** for every data element

- *Example Entity:* Vehicle
- *Example Attribute:* Vehicle_VIN
- *Example Data Format:* for Vehicle_VIN ...



One commonly used data analysis method defines data in terms of **entities, attributes and relationships**

- Definitions

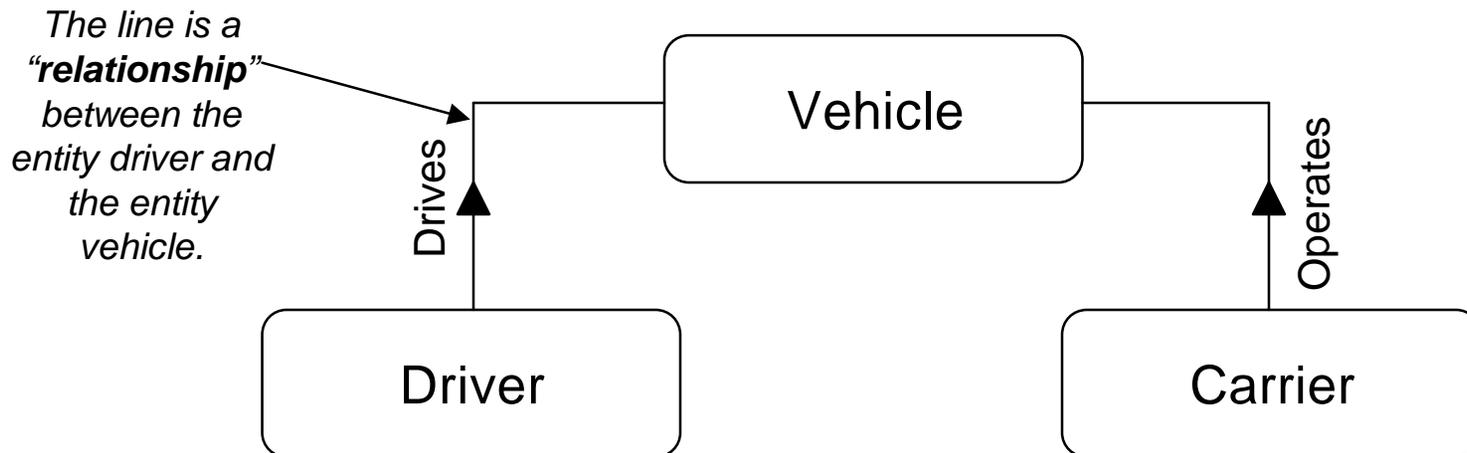
- **Entity:** Any person, place, thing, concept, or event that has meaning to an enterprise, and about which data can be stored. (Example: vehicle)
- **Attribute:** A named characteristic of an entity. Data attributes are sometimes referred to as data elements. An attribute is defined as the smallest unit of named data. (Example: Vehicle_registered_weight)
- **Relationship:** An attribute whose value is the identifier of another entity
Example: A Carrier operates one or more Vehicles. The Vehicle Identification Number (VIN) is an attribute of both the vehicle and the carrier and defines the **relationship** (or link) between them

- Examples of Vehicle Attributes: vehicle_ ...

age, annual mileage, axle_configuration, axle_spacing, weight, classification, CVSA_certification_date, factory_price, height, horsepower, make, model, model_year, odometer, oos_status, ownership, time_last_inspection, VIN, etc.

Entity Relationship Models provide a way to analyze logical relationships among entities

Example: A simplified Entity-Relationship Diagram for the Vehicle Entity



Relationship: An attribute whose value is the identifier of another entity. Relationships are read in direction of the arrow: e.g., **a driver drives a vehicle**

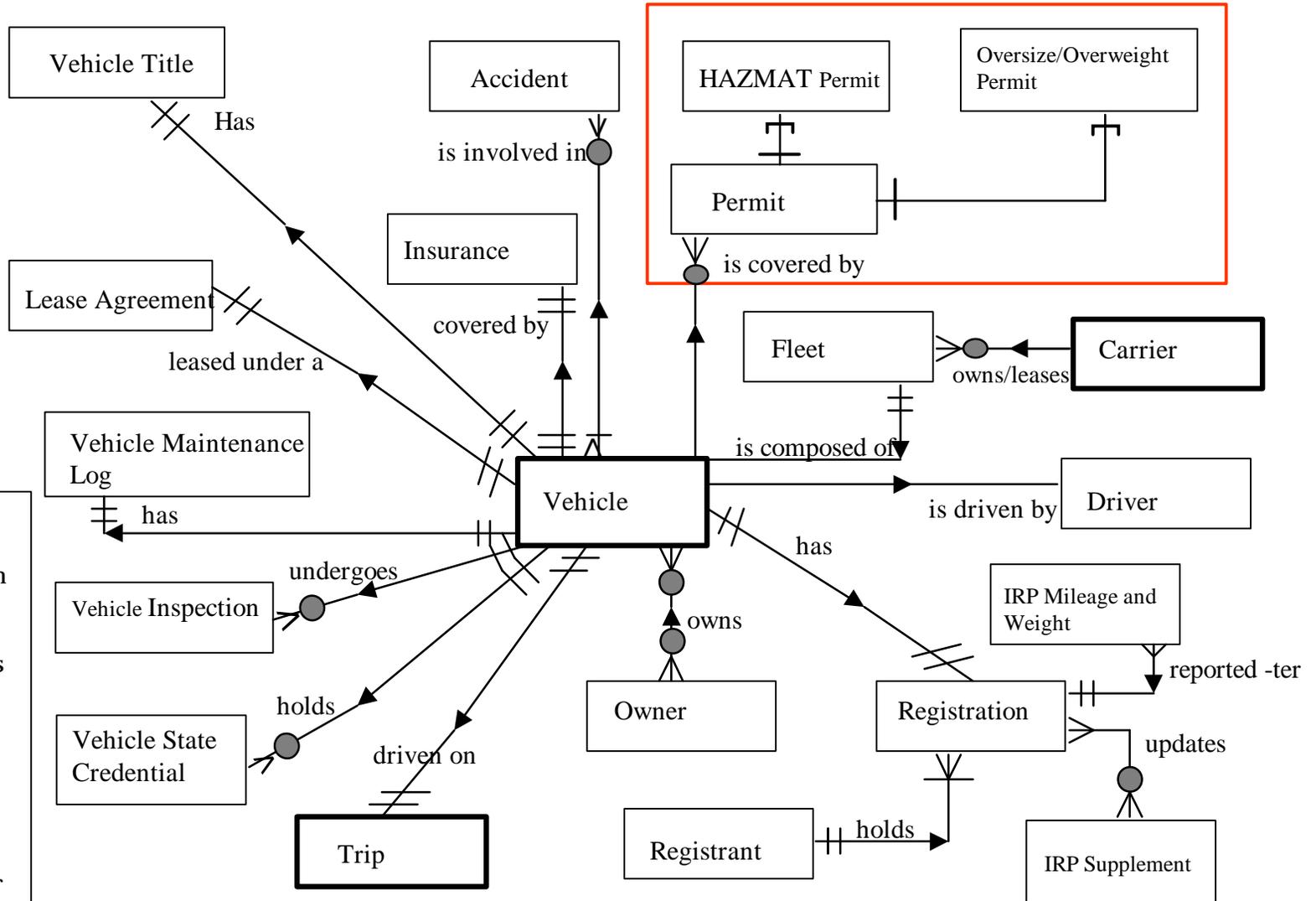
Entity-relationship (ER) diagram (ERD)

The figure shows an ERD for the vehicle entity. This diagram is an example of an analysis technique used to define and understand data. It presents a logical view of the data. This can be very helpful a step to take before defining a physical view of the data. The physical view is a design for storing data in a particular DBMS; it is a detailed design.

CVISN Data Dictionary Vehicle Entity - Relationship Diagram

This is a preliminary example of the complete ER diagram for the Vehicle entity. It is taken from the CVISN Data Dictionary, JHU/APL POR-96-6988, February 29, 1996

Legend
 The symbols near the boxes indicate the min and max number of entities in the relationship. Symbols used are zero (0), one (1), or many (∞).
 3/4 indicates that the adjacent element is a super type; ▲◆ indicates a sub-type.
 For example, a carrier owns zero to many fleets.



Data Base Management Systems (DBMS) are commercially available, general purpose products which help manage data

DBMS packages support these processes:

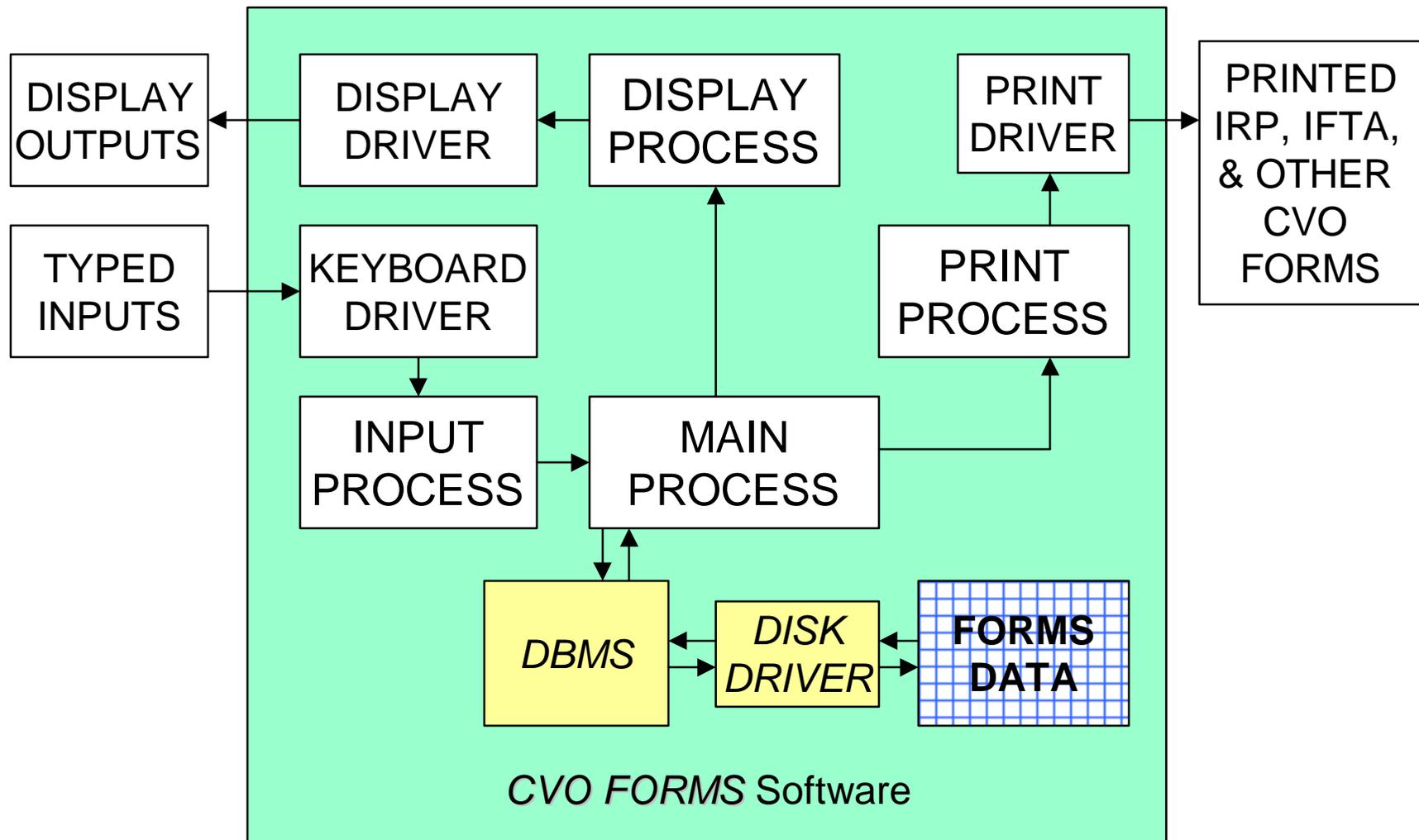
- Creating Data Structures from Data Models
- Storing & Retrieving
- Searching (Queries)
- Error & Rules Checking
- Access Control & Security
- Backup & Recovery
- Audit Trail
- Archiving

A relational database management system is a DBMS that is designed to support implementation of physical data structures based on a logical relational data model. Other types of DBMS's include hierarchical, networked, and object oriented.

DBMS products are complex software systems in their own right

- Examples of commercial DBMS products include: Microsoft Access, IBM's DB2, IDMS, Ingres, Oracle, Sybase.
- Many DBMS vendors have product versions that operate on different hardware platforms under various operating systems.
- These products can have significant initial and annual maintenance fees (thousands or tens of thousands of dollars).
- Maintenance planning needs to consider keeping up with current versions of the DBMS.

CVO FORMS Top-Level Design: Refine the Data Management Top-Level Design



A DBMS is added to the “*CVO FORMS*” software to manage the Forms Database

- In this step we refine the top-level software design.
- We add two commercial-off-the-shelf (COTS) software elements:
 - DBMS, Disk Driver
- We pull out some of the data management functions from the Main Process and allocate them to the DBMS.
- We determine that this refinement is an improvement over the previous alternative, because it saves time and has more features (by using existing software) and is more modular (usually making it more build-able, testable, & maintainable).

What if new requirements arise for the *CVO FORMS* application?

- Requirements

- Accept inputs data from a keyboard/monitor for IRP, IFTA, titling, and intrastate registration
- Print forms on a laser printer, ready for submittal to Midland



- Transfer an electronic transaction to Midland for processing

- Design Constraints

- Run on an Intel Pentium, Windows 95 Computer System

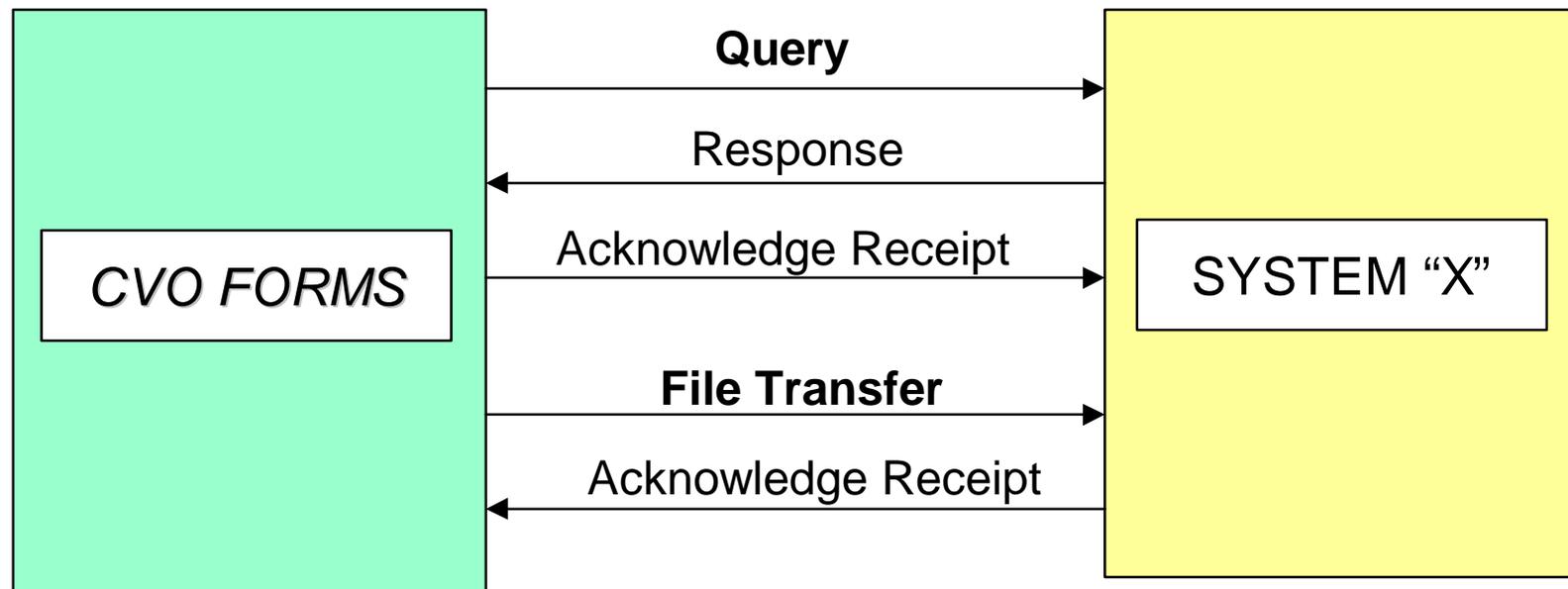
What if you got this new requirement after building the CVO Forms application to the original requirements?

A good design allows a software application to evolve over time to accommodate new requirements

- Requirements change over time for many reasons:
 - Error or omission in the original requirements
 - Changes in business needs, policies, or laws
 - Operational experience with the first version of the system encourages new ways of thinking
 - New technology
 - Changes forced by competition
 - Opportunities for cost reduction
- When developing the system initially, consider the need for establishing a solid baseline that evolves over time.
- If a system is successful and is used for many years, most of its cost is likely to occur after its initial development for maintenance and evolutionary upgrades.

To respond to this new requirement for the CVO Forms example we must think about how to connect our software with other Midland software, so we must consider external interfaces ...

System *Logical* Interfaces



From a logical perspective, systems communicate with each other by sending messages and responses to each other.

The logical interface design is concerned with what information flows across the interface, not the physical characteristics of the interface

- A message format definition defines the content of each message.

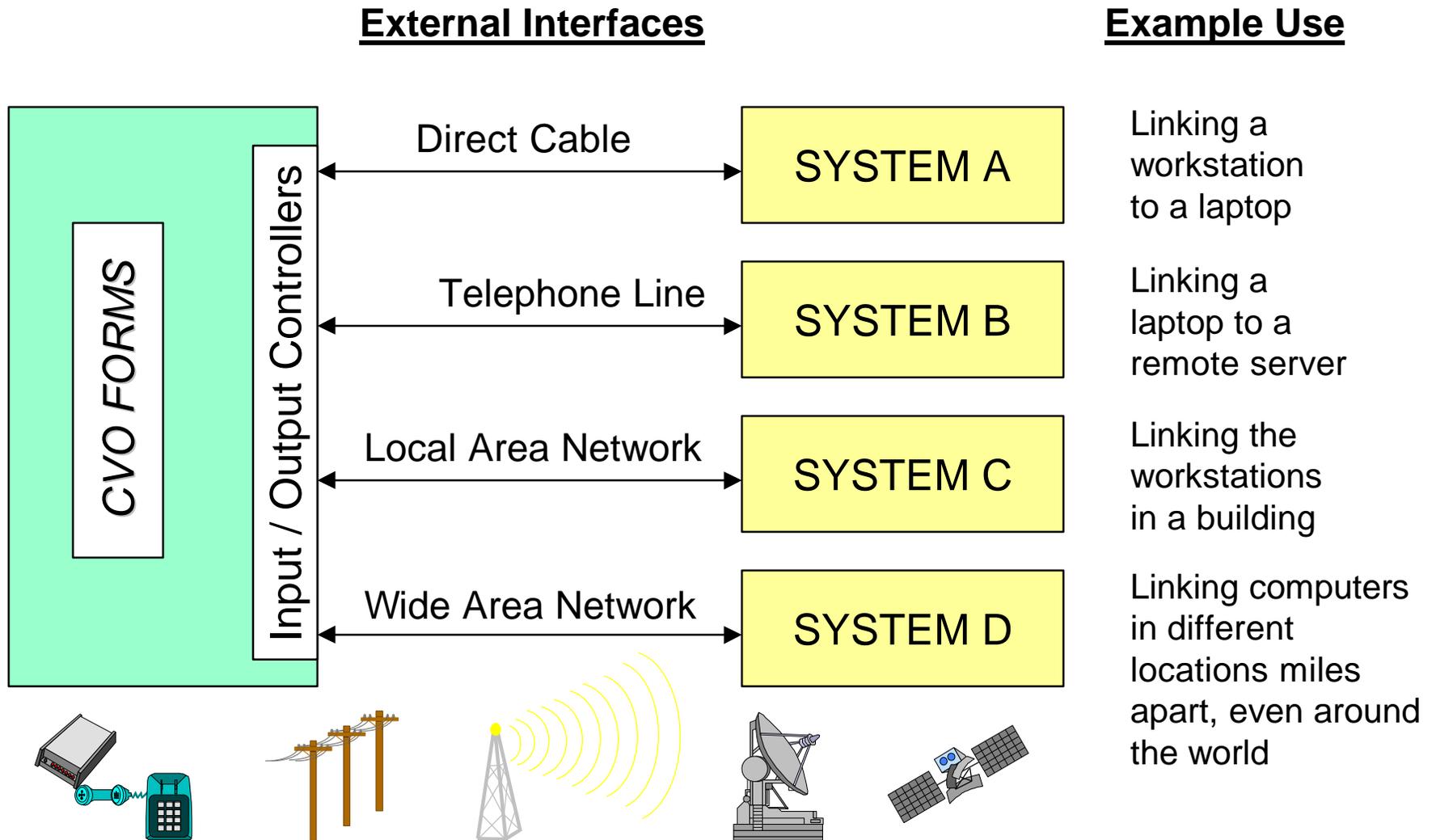
American National Standards Institute (ANSI) Electronic Data Interchange (EDI) X12 Standards are an example of one approach to defining message formats.

- A message flow *protocol* defines how messages go back and forth between systems.

Sending an EDI transaction, returning an EDI Acknowledge, and subsequently returning an EDI response transaction is an example of a protocol.

- Message formats and protocols can get quite complicated because they need to handle complex alternatives and error conditions.

System *Physical* Interfaces

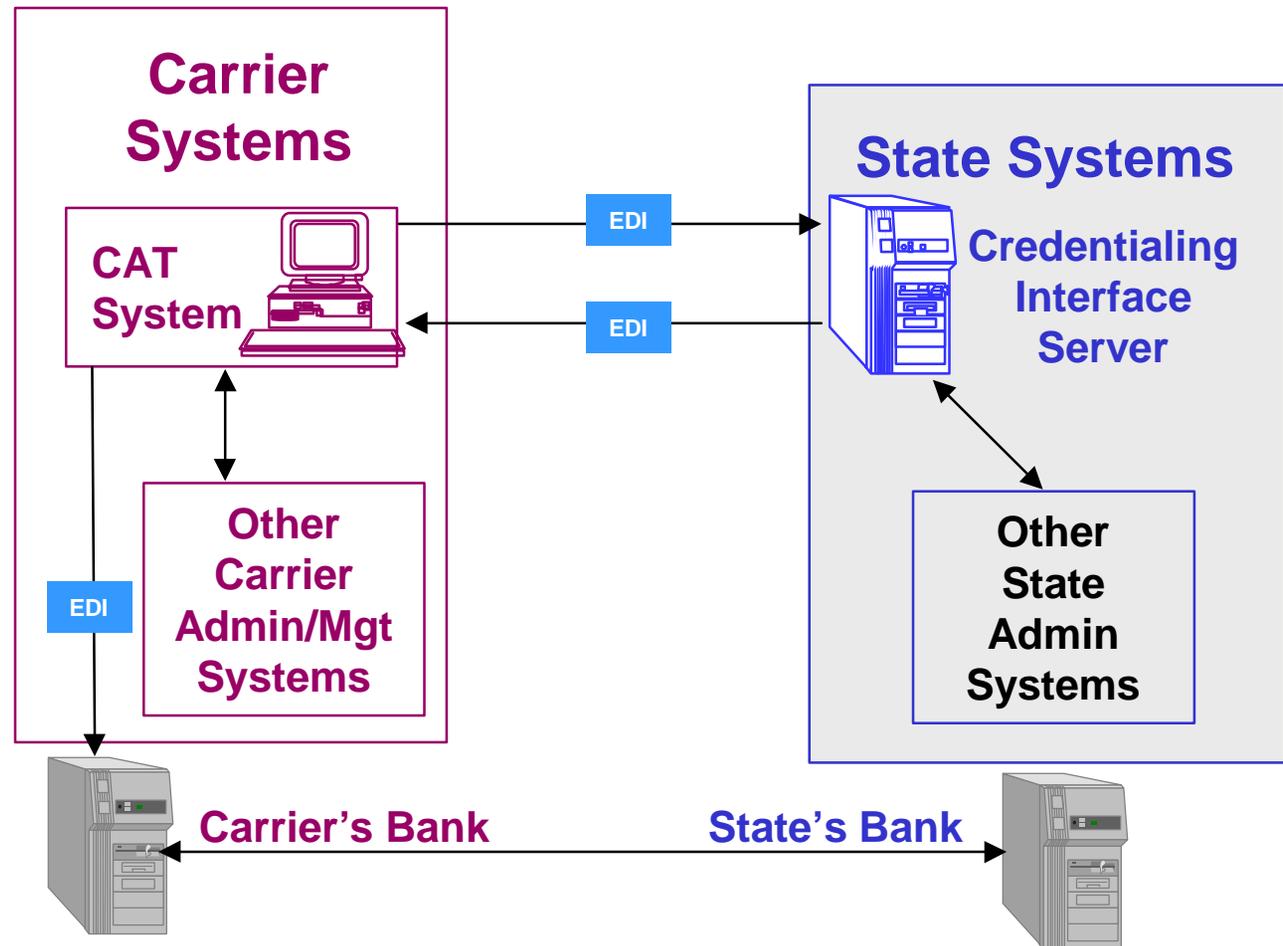


The physical interface design is concerned with the physical characteristics of the interface

- Communications and networking are a complex discipline with many sub-areas of expertise.
- Communications options have exploded over the past 1-2 decades:
 - Phone options, local area networks, cable, satellite, cellular, Internet,
- Communications costs are dropping and capacity (the amount of data that can be transferred in a unit of time) is increasing dramatically.

The Carrier Automated Transactions (CAT) system is a good example of how information systems support CVO

The CAT is one way for carriers to apply for and receive credentials electronically



The CAT software provides automated support to the carrier process of applying for CVO credentials

The Carrier Automated Transactions (CAT) software performs these functions:

- Data entry screens for credential applications & fuel tax filing
- Validate application
- Specify payment method
- Compute fees (some, not all)
- Print applications
- Translate to/from EDI transaction
- Initiate payments through banks (*future*)
- Send EDI transactions
- Receive EDI transactions
- Acknowledge EDI transactions
- Print credentials
- Archive transactions

CAT Software Demonstration

- The CAT provides a good, simple example of how information systems can be used to aid CVO.
- This demonstration will illustrate how electronic credentialing looks from the carrier's side using a commercially available CAT.
- Other information system implementation choices are also possible for carriers and states.
- ***Throughout the demonstration, watch for elements of information system design.***



Hints / Tips

The CAT software includes all the basic elements of an information system

Think about the system elements we have discussed in this module so far. Can you see how the CAT embodies or implements these elements?

- What are the Inputs? Processes? Outputs?
- Does the CAT have a User Interface? Database Management System? External Interface?
- What are the characteristics of the computer system (i.e., the hardware) that support the CAT software?



Transition to
a New Topic

Now we shift to a discussion of some management issues in developing information systems:

Issue 1: Build or Buy	Should we “Build or Buy” each system?
Issue 2: Development Process	What process should we use?
Issue 3: System Integration	How can we integrate all the systems together and test that they work?
Issue 4: Configuration Management	How can we control changes that inevitably occur?
Issue 5: Internet & Web	How can our state use the Internet & World Wide Web? <i>(addressed in Module 5)</i>

Be aware of common software development issues and have a plan to address them!

- There are some common issues that each state will need to address as part of its CVISN deployment project. These are some of the more significant technical issues. Each is discussed in more detail on the following slides (Issue 5: Internet & WWW is addressed separately in Module 5, Connecting Information Systems to each other and to Users).
- There is no single right answer to each of these that fits all states. Each state needs to plan to address these issues in the context of its situation and needs.
- Seek advice from others who have dealt with these issues.

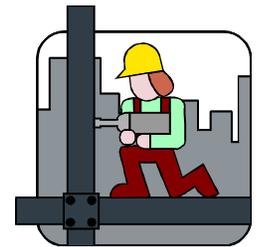
Build versus Buy

Options for implementing software:

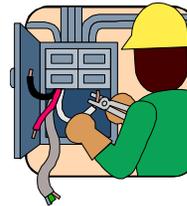
- Build everything from scratch



- Assemble commercial off-the-shelf (COTS) components and add custom software components



- Modify an existing package



- Buy a complete COTS software package and configure it



Define criteria and analyze each option in making your “Build vs. Buy” decision for each system

Approach	Cost	Schedule	Risk	Quality	Maintainability	Other
Build	Highest	Longest	Often high risk of not producing a useful product	Unknown. Depends on process & team	Highest. Must have in-house maintenance staff	Staff availability?
Assemble	High	Long	Risk that pieces may not fit together	Partially unknown	High	System integration experience?
Modify	Lower	Moderate	Known	Known. Probably similar to existing system	Lower	Quality of legacy system?
COTS	Lowest	Shortest	May not be able to meet all critical requirements	Need to make compromises to preferred business processes	Lowest. Vendor spreads costs over many customers	Available choice of packages?

- Conduct a trade-off study to decide what approach to take. Use a set of criteria to evaluate alternative approaches and select the best for you. In general, reusing existing software (either in whole or in-part) is going to be less expensive.
- The comments above are intended as food for thought about typical considerations. Specific comments can only be made on a case-by-case basis.

**Issue 2:
Development
Process**

The Waterfall Model provides a simple view of the software life cycle

Software Project Management (Planning, Organizing, Monitoring & Control)

Requirements Specification

Top-Level Design

Detailed Design

Code & Unit Test

Integration & Test

Operation & Maintenance

Software Life Cycle:
The period of time that starts when a software product is conceived and ends when the product is no longer available for use.

The waterfall model shows all the necessary activities associated with developing software. It is very helpful for understanding what must be done.

Software Project Support (System Engineering, Test & Integration, CM, QA)

However, the waterfall presents an ideal that often doesn't fit real-world problems.

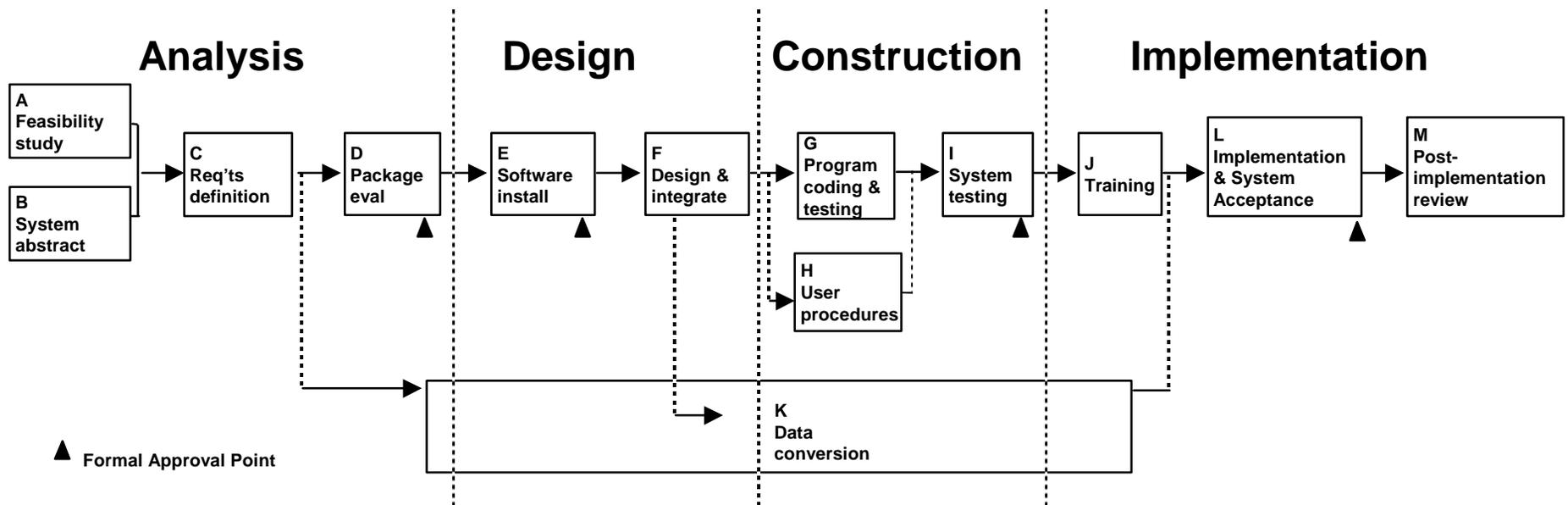
Each stage of the waterfall model has a specific purpose and result that leads to the next stage.

Requirements Specification	Defining what the software should do, mostly in business or functional terms. Also define critical design constraints (e.g., what hardware platform or operating system must be used).
Top-Level Design	Defining what the major software elements are and how they interface with each other. Defining the top-level database structure and user interface structure. (Also sometimes referred to as preliminary design or software architecture).
Detailed Design	Defining what the step-by-step algorithms are for each software unit. Defining the details of each data element in the database. Completing the user interface design.
Code & Unit Test	Writing code and testing (possibly through manual walk-through) the smallest level units.
Integration & Test	Bringing the software units together in increasingly bigger groups and testing that they work as an integrated whole.
Operation & Maintenance	Using the system, fixing previously undetected errors, and evolving it to meet changing requirements.

Using COTS software can save time and money if good packages exist in the marketplace

- Requirements definition is still necessary to help choose among alternative commercial products. But the required level-of-detail is not as great as when developing software from scratch.
- The Design, Code & Unit Test, and Integration & Test stages are all very different and reduced in scope. They still exist because usually some package configuration or modification is necessary. Also, development of interfacing software or modification of existing legacy systems may be required.
- You may also have to change your business practices to accommodate the package. Often packages inherently bring with them the lessons learned from other organizations. This can sometimes be a good opportunity to improve your practices.

A life cycle for commercial off-the-shelf software (COTS) focuses on picking, configuring & customizing a product



- This example of a life cycle for packaged software reflects the fact that you aren't building from scratch.
- The basic tasks are the same as in the waterfall, but the detailed sequence of steps and detailed methods used in each step are very different.

You must pick a good life cycle model for your situation

Various models have been developed to accommodate shortcomings of the waterfall in a range of situations.

Alternative life cycle models involve:

- Feedback from later stages to earlier stages
- Breaking development into increments and iterating the cycle
- Changing the order of some steps
- Starting from an existing software base vs. from scratch
- Varying the amount of formal structure to accommodate the size of the project (e.g., using more (or less) formal documentation).
- Using computer aided software engineering tools

Many alternative life cycle models are commonly used

- Waterfall
- Waterfall derivatives
 - overlap
 - feedback
 - build
- Code and Fix (Wing It!)
- Evolutionary
- COTS Package
- Quick React
- Spiral
- Automated Code Generation
- Rapid Prototyping
- Others ...

- Many alternative life cycles have been proposed in technical literature.
- Private companies have developed hundreds of variations of life cycles and development methodologies.
- The reference section for this module contains more examples.

**Issue 3:
System
Integration**

Integration and test takes pieces of the system & systematically brings them together into a working whole

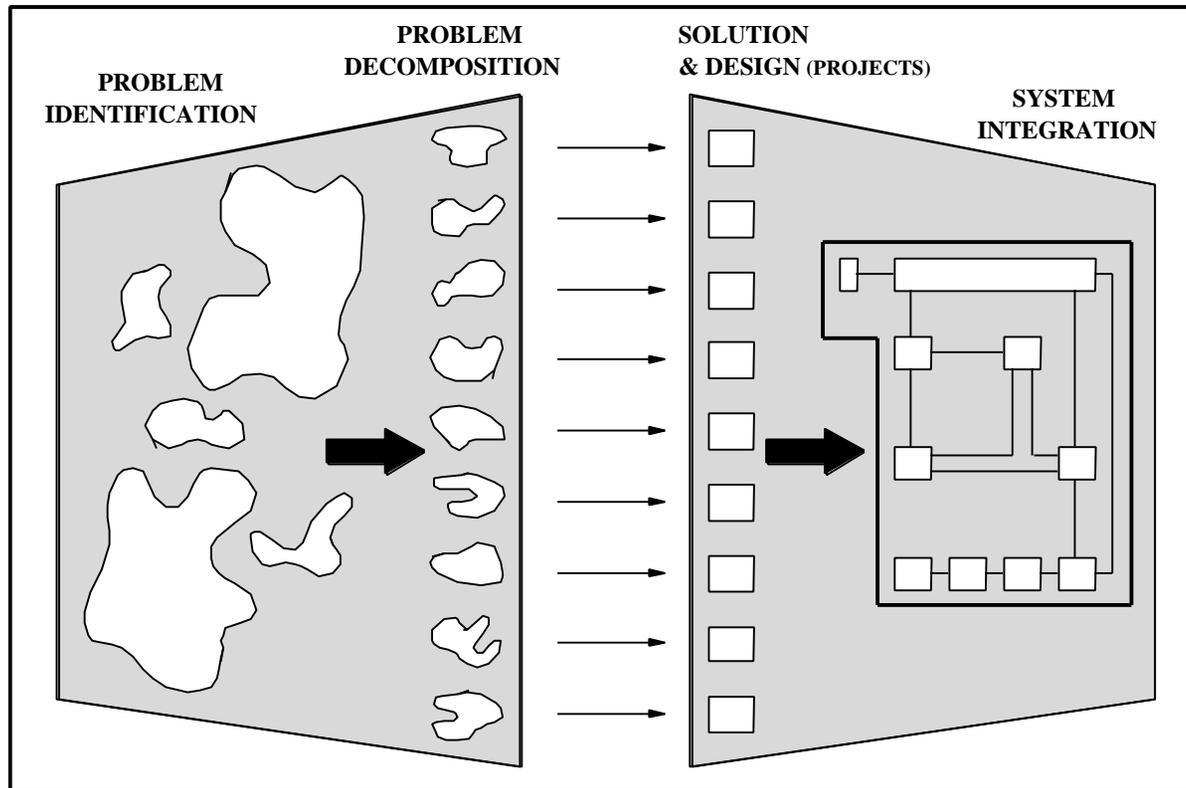
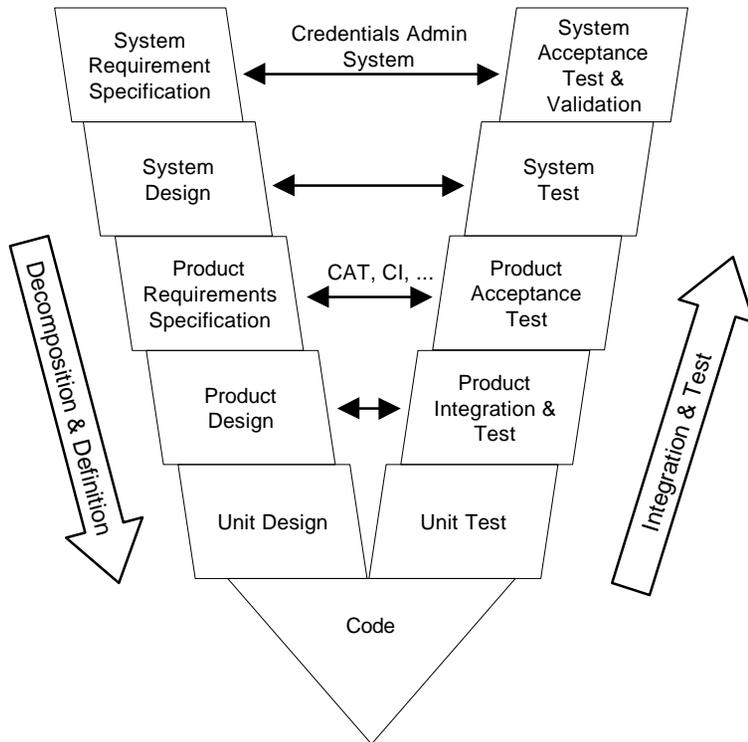


Figure from *An Effective Approach to System Integration: A Comprehensive Checklist* by Scott A. Hyer. See reference materials.

Complex problems must be decomposed into many smaller problems which resemble problems with known or workable solutions. This decomposition brings about many small solutions which calls for an integration activity to bring about a satisfactory system solution.

The integration & test effort is significant.
Planning & preparation for it should begin early in the life cycle



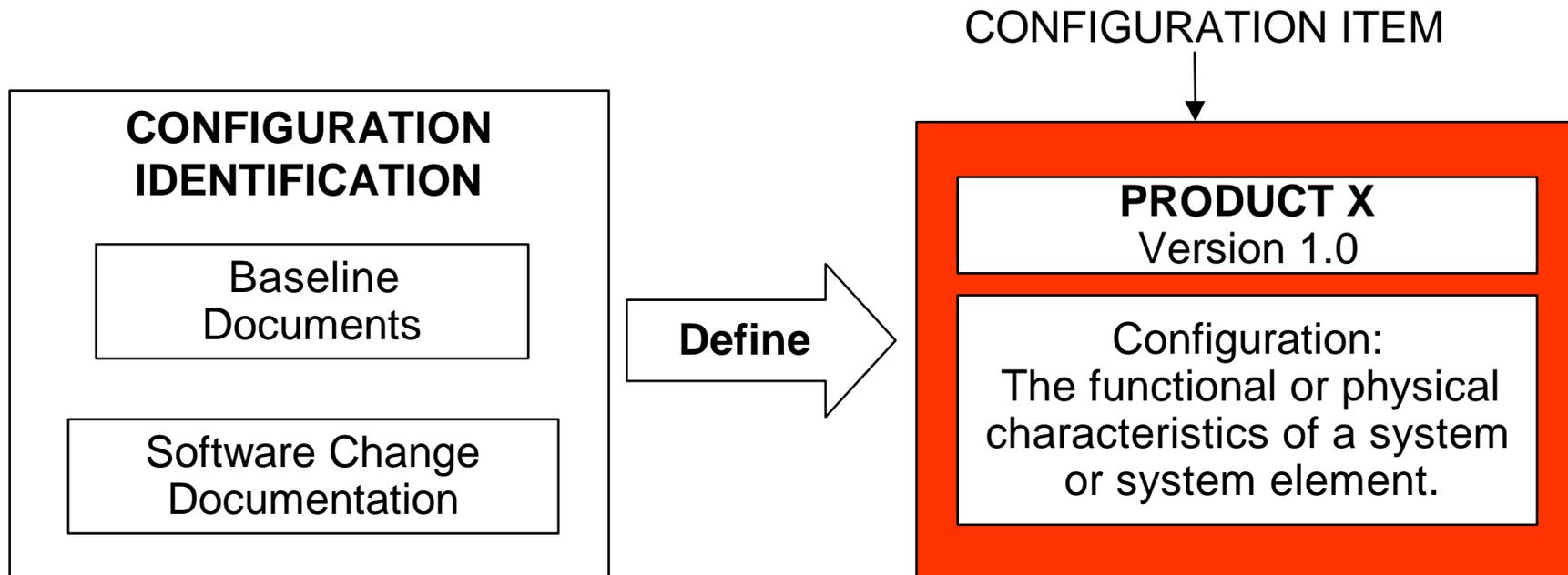
Use a multi-level test approach like the one illustrated in this “V” diagram.

- Use a defined integration & testing methodology. Carefully plan and document. Don't just wing it!
- Integration with legacy systems is often the most difficult problem.
- Acquiring and configuring networks & communications is a large task. Security issues can be tough to solve.
- Don't forget to consider required changes to business practices to use the new system.
- Devote adequate time and resources. Integration & test can take 20% or more of total system development time & cost.
- Start early. Pick a basic capability, then develop & integrate early versions of products to work through integration problems.

**Issue 4:
Configuration
Management**

Configuration Management (CM):

Defining the characteristics of hardware and software items and controlling changes to these items



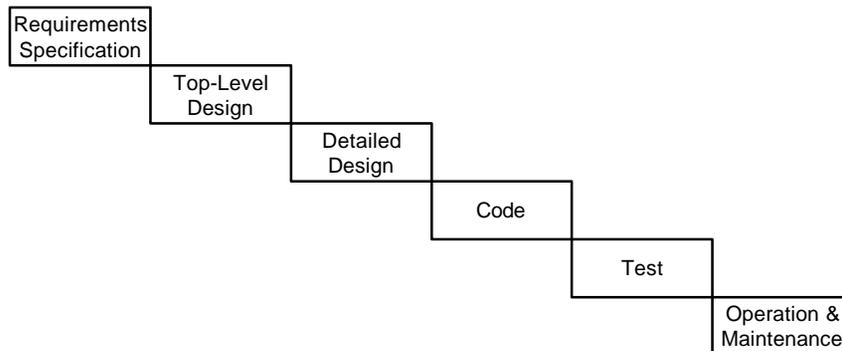
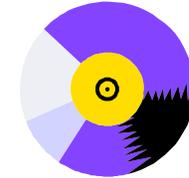
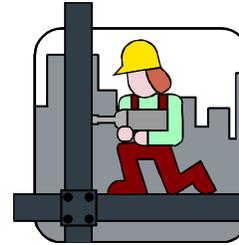
Baseline : The documented characteristics of some aspect of a project or system, formally approved by a Configuration Control Board (CCB)

A good CM process will encourage precise, effective technical communication among all development participants. It can help to avoid many wasted hours due to miscommunication and disorganization

- **CM activities include ...**
 - Identifying the configuration management items
 - Controlling changes to configuration items
 - Establishing and maintaining status of changes and implementations
 - Allowing approval of all changes
- **Keys to successful CM**
 - Have a good plan
 - Train everyone in their role
 - Be very organized about documentation & record keeping
 - Strike a good balance between control & responsiveness
- **Configuration management (CM)** is a discipline applying technical and administrative direction and surveillance to identify and document characteristics of a configuration item.
- A **configuration item** is the hardware, software, or design that is under control.
- The Configuration Management process will be administered with the aid of a **Configuration Management System**. This system will be used to identify, control, audit, and review all configuration items. The process is managed by configuration control boards.
- **Change control** is an element of CM consisting of evaluating, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification.
- The **configuration control board (CCB)** is a team of technical and program management representatives who are responsible for evaluating and recommending corrective actions to system or software discrepancies.
- A **Change Request Form** is a form used to report problems or request changes to a configuration item.

How has your state acquired or developed information systems in the past?

Discuss examples in your state of how the “build vs. buy” decision was made. What criteria were used?



What types of life cycle models have been used in your state? Consider both successful and unsuccessful approaches

What “lessons learned” have you gleaned from these experiences?



Many alternative life cycle models are commonly used. Are you aware of any that have been used in your state?

Some questions to prompt your thinking:

- What factors did you consider in the “build vs. buy decision?”
 - Cost
 - Schedule
 - Risk
 - Quality
 - Control
 - Technical
 - Other?
- What life cycle model was used?
- How did your model differ from the classic waterfall model?
- Did your model involve iteration?
- Did your model develop items incrementally? If so, were all the requirements developed up front? Was the design completed up front?
- Does your information systems department have a well structured model that they follow?

Life Cycle Models

As a reminder, some types of models are:

- Waterfall
- Waterfall derivatives
 - overlap
 - feedback
 - build
- Code and Fix (Wing It!)
- Evolutionary
- COTS Package
- Quick React
- Spiral
- Automated Code Generation
- Rapid Prototyping
- Others ...

Recap & Questions

Learning objectives:

- Explain the components and architecture of an information system and how data flows in that system
- Understand the information systems development process and some related issues

Any questions?