Software Engineering I CSC-382

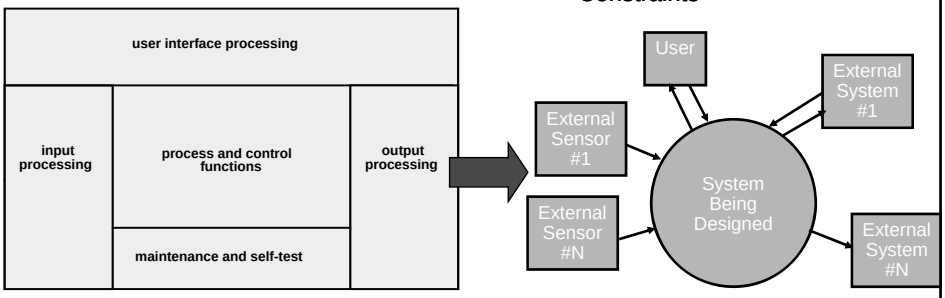
Final Review

Review of Flow of Analysis-to-Design Process I

Functional Requirements

Non-Functional Requirements

Constraints



System Context Diagram

Software Context Diagram

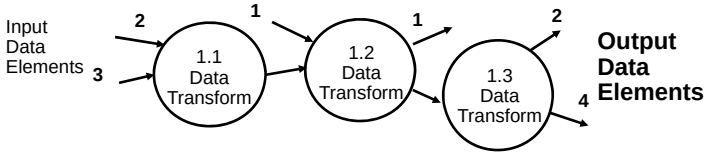
Recall the Tools We Used for the Remaining Analysis Modeling

- **Structured Analysis-based Techniques**
  - Use Data Flow Diagrams, Control Flow Diagrams for the Flow Models and State Transition Diagrams for the Behavior Models
  - Finish with process narratives (PSPECS) and control specifications (CSPECS) to define low level requirements.
- **Object Oriented Analysis-based Techniques**
  - Derive the details of Class Models (using CRC Cards) and hierarchies (Class Diagrams)

Modeling via Structured Analysis

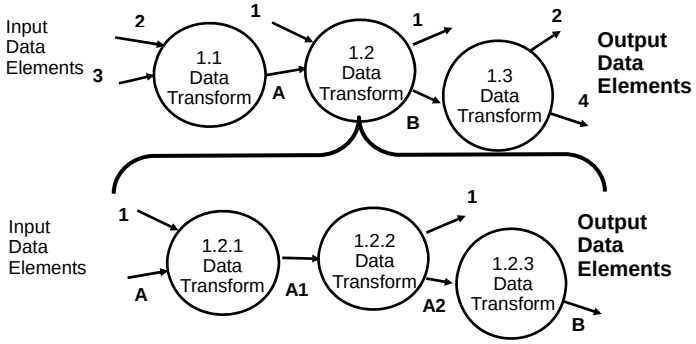
- **Structured Analysis** view a system as ***a sequence of transformations*** operating on the input data and leading to the final outputs
  - Represents how data objects are transformed at they move through the system
  - Starts with the Context Diagram
  - Continue with Data Flow Diagrams to define all the data transformations
  - Down to the lowest level uses Process Specifications (PSPECS) (narratives) to define the data transformations required at the lowest desired level of modeling

Review of Structured Analysis Process – Develop DFDs

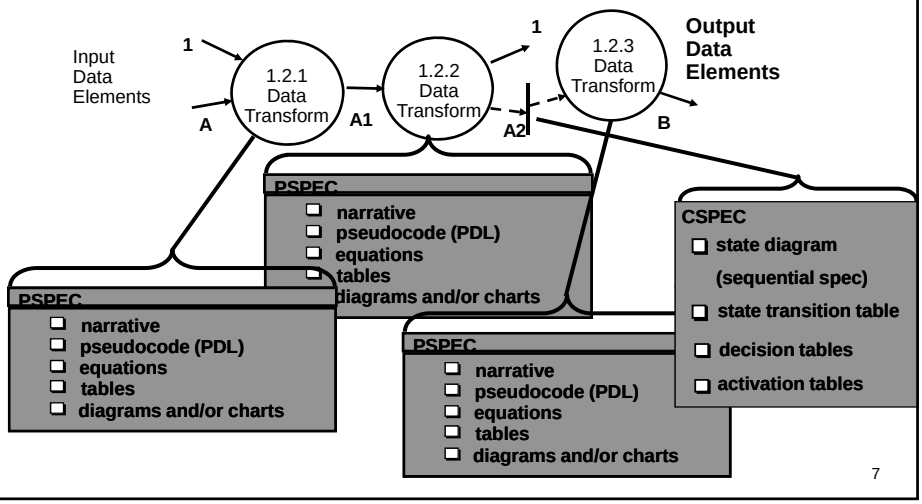


- Elements of Data Flow Model
  - **Input Data** – the inputs to the current transformation
  - **Data Transformation** – the processing to be executed on input data
  - **Output Data** – the outputs from the current transformation
- Context diagram is the level '0' Data Flow
  - The data flows are generated in a top-down process

Review of Structured Analysis Process – Flow down DFDs as details needed



Review of Structured Analysis Process –
Develop PSPECs for all lowest level transforms



Review of Structured Analysis Process–
Define Component Partitioning and Map to an Architecture

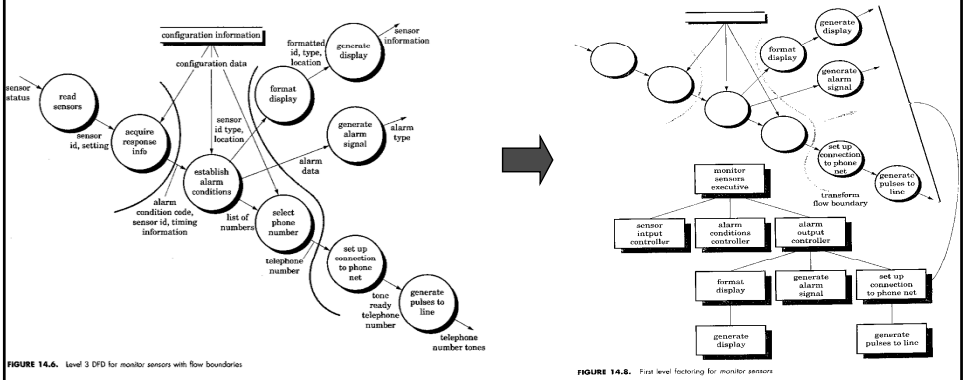


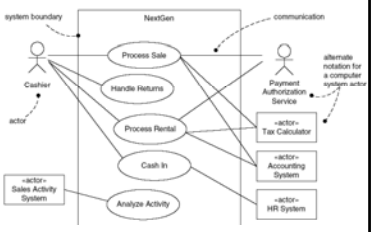
FIGURE 14.6. Level 3 DFD for *monitor sensors* with flow boundaries

FIGURE 14.8. First level factoring for *monitor sensors*

Analysis Modeling via OOA

- Object Oriented Analysis views a system as a ***collection of entities*** that communicate to each other thru messages
  - These messages request the various other objects to perform some function or task
- Identify **classes** by examining the problem statement
  - We will show a CRC-based method for making this a little easier
- Identify the **attributes** of each class
- Identify **operations** that manipulate the attributes
- Later define **inheritances** and **aggregations** of these classes.
- Also later on define **associations** of the classes based on how they collaborate to accomplish the required functionality

Review of Object Oriented Analysis – Begin with Use Cases



- **Actors** represent roles people or
- devices play as the system functions
- Three types of actors:
  1. **Users**
  2. **Administrators**
  3. **External Programs and Devices**
- Actors have these two common features:
  1. **External** to the application
  2. They take **initiative, stimulate and interact** with our system

Use Case Name:
Primary Actor:
Description of Usage:

Review of Object Oriented Analysis – Develop Class List

- Good Object candidates often represent:
  - The **work** the system performs
  - **Things** directly affected by or connected to the application
  - **Information** that flows thru the software
  - **Decision** making, **control**, and **coordination** activities
  - **Structures** and **groups** of other lower level objects
  - Representations of real-world things the system knows something about
- It can also help thinking in terms of some common ***stereotypes*** of objects to get ideas

Actor Classes:
•People
•Organizations

Interface Classes:
•Screens
•Menus

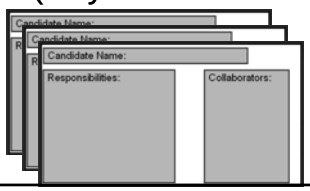
Business Classes:
•Places
•Things
•Concepts
•Events

Report Classes:
•Printed
•Electronic

Review of Object Oriented Analysis – Develop CRC Cards

- CRC stands for: Class, Responsibility and Collaborators
  - **Class**: An object is a person, place, thing, event, or concept
  - **Responsibility**: Anything that a class knows or does
  - **Collaborator**: A class that another class needs to accomplish its purpose
- How:

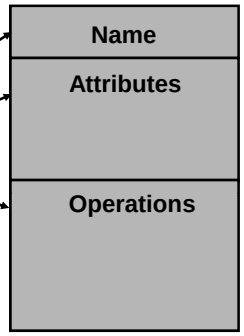
1. Walk thru the Use Case scenarios (normal scenarios first)
2. Group the Cards (maybe use Collaboration Drawing)



Review of Object Oriented Analysis – Develop Class Diagrams (a)

First define the simple Class diagram

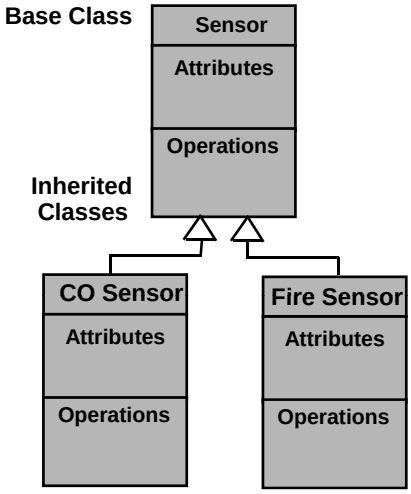
The three elements of a class definition are derived directly from the CRC cards:



Review of Object Oriented Analysis – Add Inheretances to Class Diagrams

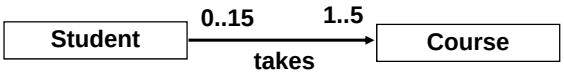
Then determine any inheritances:

- Look for common categories for the objects
- Look for common roles that all the objects play
  - Try to combine shared responsibilities from multiple classes into unified interfaces
- Then can define an abstract class to hold these shared responsibilities
  - Makes it easier to ensure that other classes can interface smoothly with these related classes.

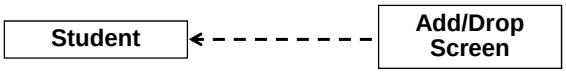


Review of Object Oriented Analysis –
Add relationships to Class Diagrams

- Then determine any **Relationships** that may exist between classes
  1. **Association**
     - A persistent relationship between objects

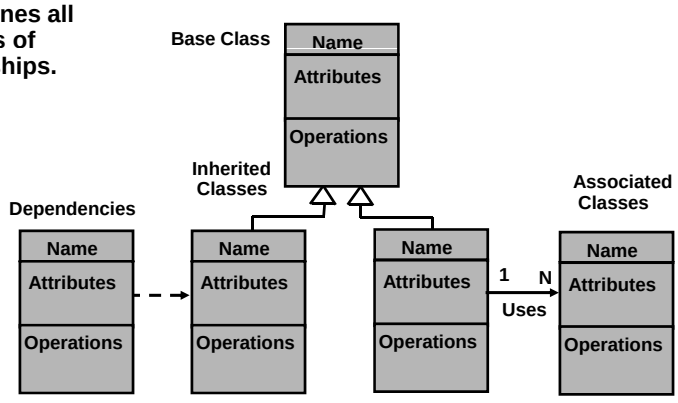


  2. **Aggregation** (and a related concept called Composition)
     - Another persistent relationship
  3. **Dependency**
     - A transitory relationship between objects



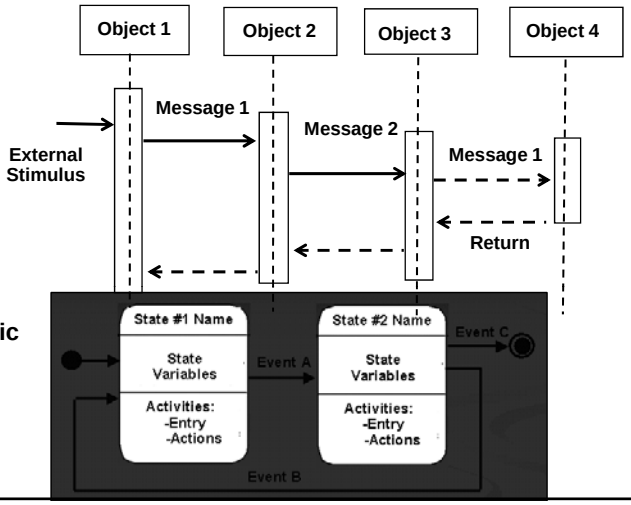
Review of Object Oriented Analysis –
Example Complete Class Diagram

…Combines all the types of relationships.



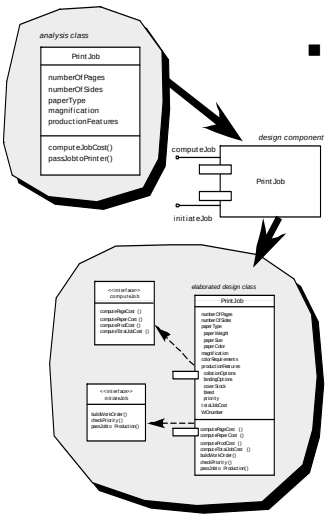
Review of Object Oriented Analysis – Develop Sequence Charts for Objects

…Now model all inter-class messages using sequence diags.

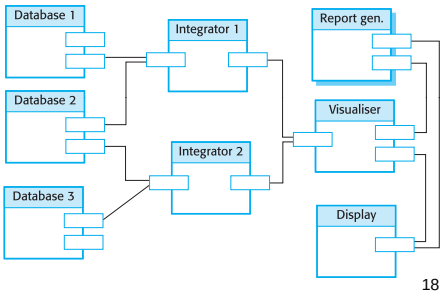


and model the logic to handle the messages using state charts.

Review of Object Oriented Analysis – Define Component Partitioning



- A component contains a set of collaborating classes
  - Recall from our CRC card efforts we defined the collaborations and also the collaboration diagram

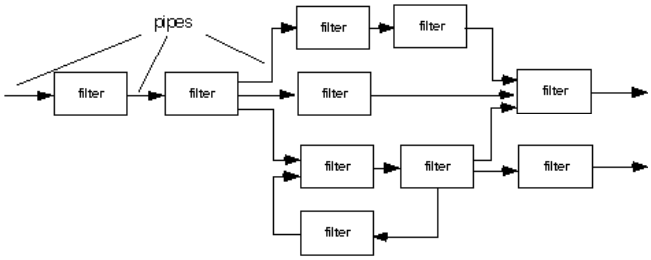


For Both Methods we Must now Map onto a Common Architectural Style

Data Flow Systems	Virtual Machines
Call and Return Systems	Data-centered Systems (repositories)
Independent Components	Process Control Systems
Layered Hierarchy (Abstract machine)	

From: Shaw & Garlan

Architectural Styles – Data Flow Architecture



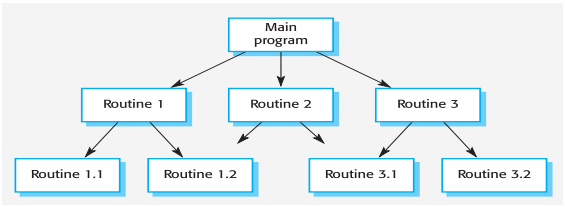
(a) pipes and filters

(b) batch sequential

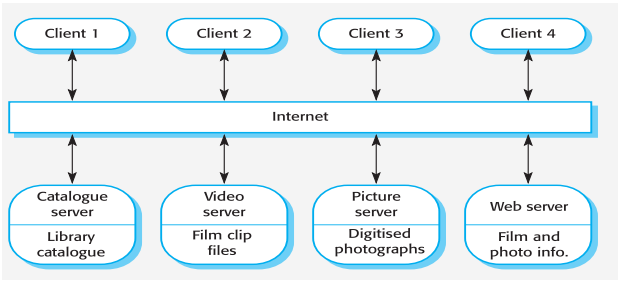
In batch sequential, data is only passed onto the next Filter when it is completely processed by the previous filter

Architectural Styles – Call-return Architecture

1. Main program-sub-program

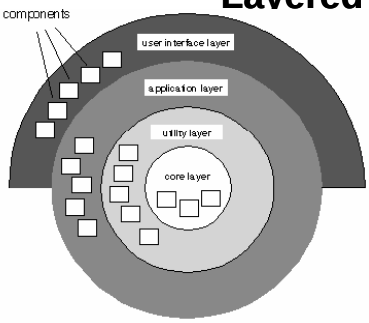


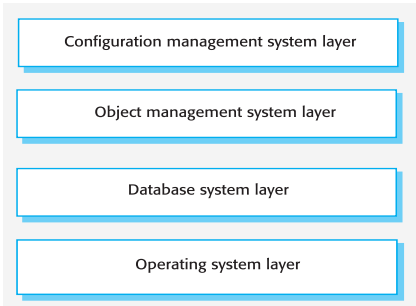
2. Remote Procedure Call (client server)



From: Sommerville

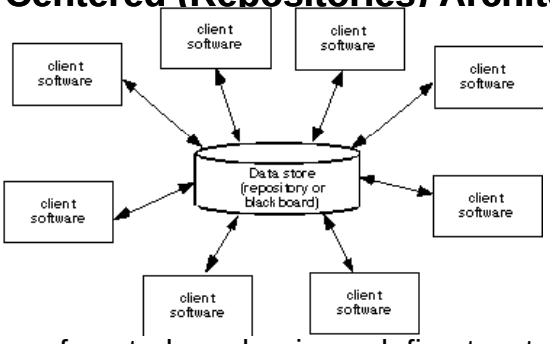
Architectural Styles – Lavered Architecture





- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Layers only communicate with nearest neighbor layers
- However, often artificial to structure systems in this way, but systems like protocol stacks and Operating Systems fit it nicely.

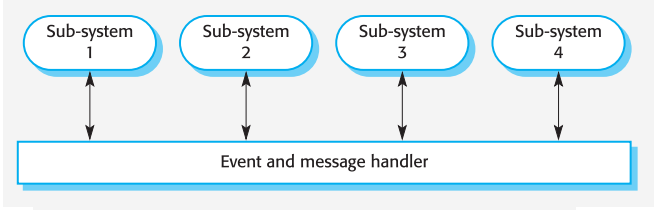
Architectural Styles – Data-Centered (Repositories) Architecture



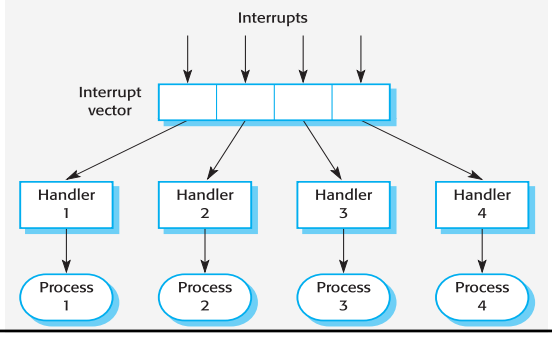
- Two types of control mechanisms define two types of systems
  - In **Database** systems, the type of transaction entered into the data store triggers which process to execute
  - In **Blackboard** systems, the current state of the central data store triggers the process to execute.

Architectural Styles – Event-driven Architecture

- Broadcast:



- Interrupt:



Review of Structured Analysis Process –
For Detailed Design We Carefully Define all PSECS and Create any Additional Transforms

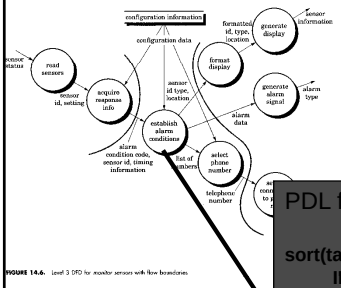


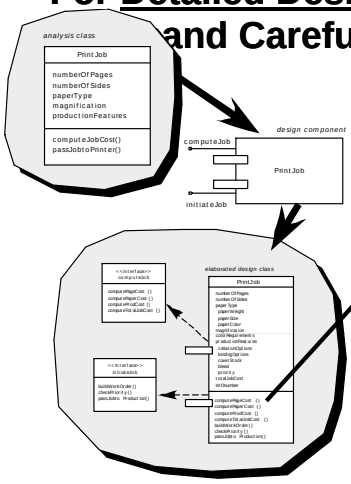
FIGURE 14.6. Level 3 DFD for monitor sensors with flow boundaries

Develop PDL definitions for every transform

PDL for each Transform

```
sort(table, number of items)
    IF there are any items
        DO UNTIL no items are interchanged
            DO FOR each pair of items in the table (1-2, 2-3, 3-4, etc.)
                IF first item of pair is greater than second item of pair
                    Interchange the two items
                ENDIF
            ENDDO
        ENDDO
    ENDIF
```

Review of Object Oriented Analysis –
For Detailed Design Define Additional Methods and Carefully define each method



Develop OCL definitions for every method in each class

```
context PrintJob::validate(upperCostBound : Integer,
custDeliveryReq :
    Integer)
  pre:  upperCostBound > 0
      and custDeliveryReq > 0
      and self.jobAuthorization = 'no'
  post: if self.totalJobCost <= upperCostBound
        and self.deliveryDate <= custDeliveryReq
      then
        self.jobAuthorization = 'yes'
      endif
```

What does a Software Tester Do?

- "The goal of a software tester is to find bugs."
- …
- "The goal of a software tester is to find bugs, and find them as early as possible."
- …
- "The goal of a software tester is to find bugs, and find them as early as possible, and make sure they get fixed."

Make Reviews a Priority to Ease Testing

- For smoothest testing team should use formal reviews throughout the project
  - These are team reviews of all the work products
  - IBM study showed 85% of errors can be removed thru these reviews
  - Again much easier to recover from errors at this stage since most of work still to be done, rather than at code testing where all work would need to be repeated
- This approach ensures you are not trying to 'test in quality'
  - The quality of the product is ensured throughout the entire development process, and instilled thru the methods we have defined to date.

A bug occurs when one of the following is true:

1. The software ***does not do something*** that the product specification said ***it should***.
2. The software ***does something*** that the product specification says ***it should not do***.
3. The software ***does something*** that the product ***specification does not mention***.
4. The software ***does not do something*** that the product ***specification does not mention, but it should.***
5. The software is difficult to understand, hard to use, slow, or ***will be considered not quite right by the end user***.

Patton

Strategic Testing Issues

- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes “rapid cycle testing.”
- Build “robust” software that is designed to test itself
- Use effective formal technical reviews as a filter prior to testing
- Conduct formal technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

Some Handy Testing Definitions

- Verification versus Validation
  - **Verification** ensures the software we are developing is correctly implementing the functions we identified as required
    - The testing methods we will be discussing are geared towards Verification.
  - **Validation** ensures that the functionality we identified as required is actually traceable to a customer requirement

Software Testing Axioms

- It is impossible to test a program completely
  - Even a simple calculator program would require an infinite number of tests if every possible combination tested.
- Software Testing is a risk based exercise
  - As soon as you decide not to test every possible path you have taken on risk.
  - Key role of the tester is to reduce the infinite possible set into a finite set of manageable tests that will ***minimize*** risk

Taxonomy of Testing

- Black Box versus White Box testing
  - In black box testing the tester only knows what the product is supposed to do, but he cannot look in and see how it operates
  - In white box testing, the tester has access to the code and can examine it to decide how to test it
    - White box testing can be prone to accidentally making the tests match what the tester sees, rather than what was intended.

Patton

Taxonomy of Testing II

- Static versus Dynamic Testing
  - Static testing refers to testing when to code is not running
    - Involves specification reviews, code walk-thrus, etc.
    - Dynamic testing is the traditional run the code thru cases
    - E.g. Testing the specification is Static testing

Patton