



# COMP 5700/6700/6706

## Software Process

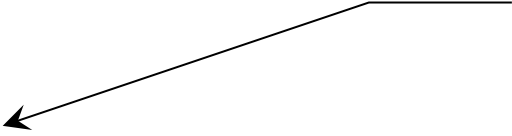
Spring 2016  
David Umphress

**Architecture**

- Lesson: Architecture
- Strategic Outcomes:
  - To understand the role and process of early component identification
- Tactical Outcomes:
  - To comprehend the role and purpose of a architecture
  - To comprehend the role and purpose of CRC cards
  - To be able to perform an architectural design on a sample problem
- Support material:
  - Reading: OO Design with CRC Cards
- Instant take-aways:
  - CRC cards
- Bookshelf items
  - Shaw, M. and D. Garlan. 1996. Software Architecture. Prentice-Hall.
  - Gamma, E., et al. 1995. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley
  - Swartout, W. and R. Balzer. 1982. On the Inevitable Interwinning of Specification and Implementation. *Communications of the ACM*, 25, 7, 438-440

# Syllabus

- Software engineering raison d'être
- Process foundations
- Common process elements
- Construction
- Reviews
- Refactoring
- Analysis
- Architecture
- Estimation
- Scheduling
- Integration
- Repatterning
- Measurements
- Process redux
- Process descriptions\*
- Infrastructure\*
- Retrospective

- 
- **Architecture defined**
  - **Components**
    - **Decisions**
    - **Scenario-component map**
    - **Techniques**
      - **CRC**
      - **Wiring**
      - **Class**

# Our Process So Far

## Minimal Guiding Indicators

Goal	Indicator
Cost:	Don't care
Schedule:	Don't care
Performance:	
Product:	
NFR:	Don't care
FR:	75% reqmt id
Process:	pain < value

## Minimal Sufficient Activities

### Engineering Activities

Envision  
Analyze  
Synthesize  
Architect  
Articulate

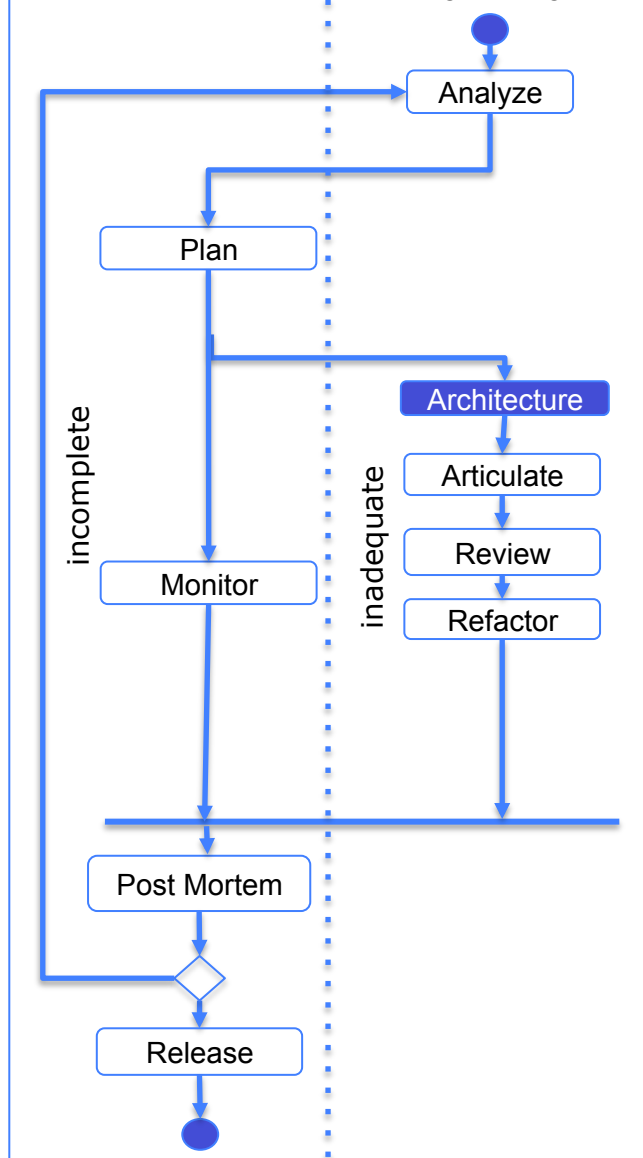
Interpret

### Operational Activities

Plan

Monitor  
Post Mortem  
Release

## Minimal Viable Process



## Minimal Effective Practice

MSA	MEP
Analyze	scenarios/user stories
Synthesize	????
Articulate	TDD
Review	Test code coverage
Refactor	Bad smells
Monitor	Ad hoc
Post Mortem	Ad hoc
Release	Eclipse zip spreadsheets

How do you determine the "how" of your solution?

How do you determine the magnitude of the development effort?

# COMP5700/6700/6706 Goal Process

## Minimal Guiding Indicators

Goal	Indicator
Cost:	None
Schedule:	PV/EV > .75
Performance:	
Product:	none
NFR:	none
FR:	100% BVA
Process:	pain < value

## Minimal Sufficient Activities

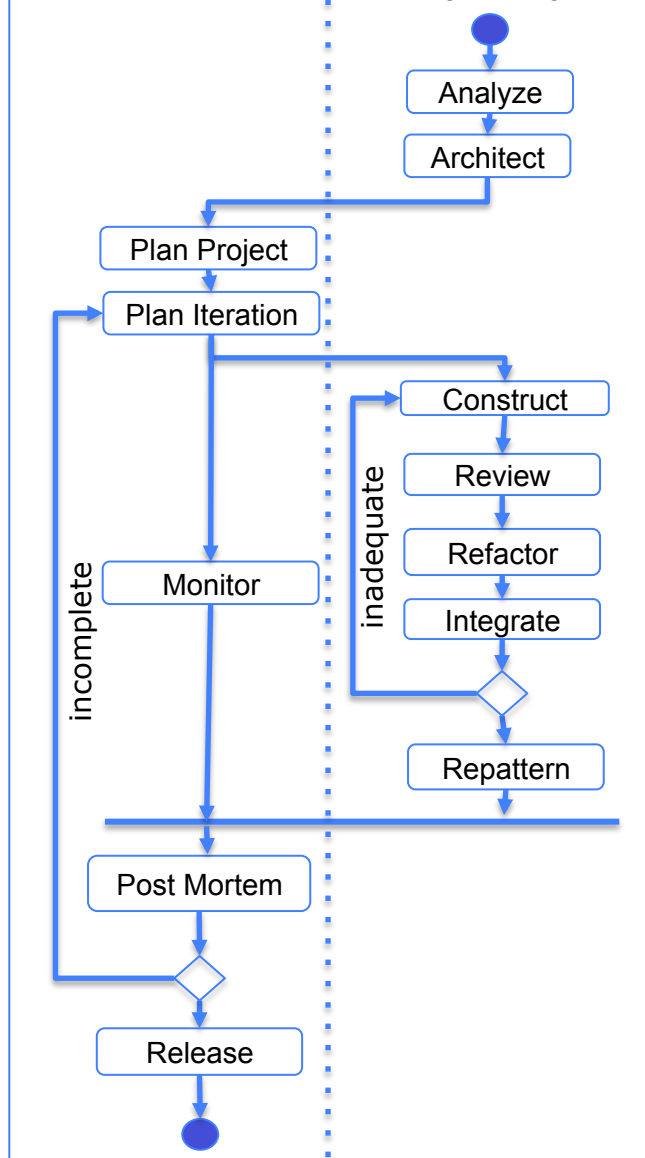
### Engineering Activities

Envision  
 Analyze  
 Synthesize  
 Architect  
 Articulate  
 Construct  
 Refactor  
 Interpret  
 Review  
 Integrate  
 Repattern

### Operational Activities

Plan  
 Plan project  
 Plan iteration  
 Monitor  
 Release

## Minimal Viable Process

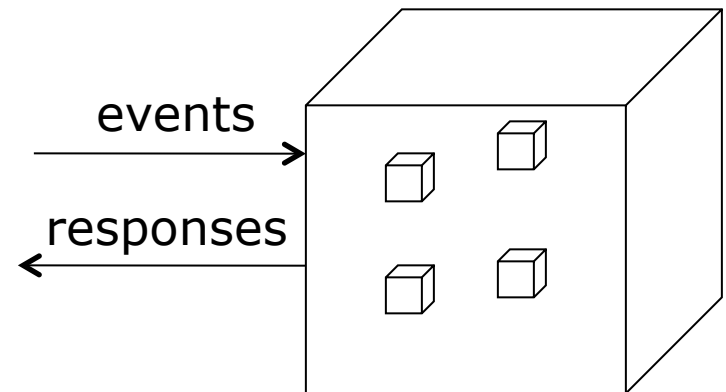


## Minimal Effective Practice

MSA	MEP
Analyze	Scenarios
Architect	CRC
Plan Project	Component-based estimation
Plan Iteration	Component-iteration map
Construct	TDD
Review	Review checklist Test code coverage
Refactor	Ad hoc sniffing
Integrate	Ad hoc
Repattern	Ad hoc
Monitor	Time log Change log Burndown
Post Mortem	PV/EV
Release	Eclipse zip spreadsheets

# Scoping

- PCSE takes a 2-prong approach:
  - identify interaction from a black box perspective
    - provides validation of feature set selection
  - identify parts within the black box at a limited level of abstraction ... an "architecture"
    - provides basis for estimation, task identification, and scheduling
    - articulated with CRC cards, wiring diagrams, class diagrams, etc.



NB: architecture = high-level design for our discussion purposes, but could be a true architecture

# PCSE Architecture ...

- ... is a preliminary forecast of the end solution using the limited information that is available in the early stages of a project.
- ... is a means of determining initial scope and effort (and, therefore, resources, cost, schedule)
- ... is important
  - 80/20 rule: 80% of the cost is determined by decisions made within the first 20% of the project effort
- ... requires a careful balance of resources
  - enough to envision needed components
  - not a locked-in design ... because the design is likely to evolve
- ... is an instantiation of E-S-A-I activities at a level of detail sufficient to plan and to begin building

# PCSE Architecture – Prong 2

- Components

- Concept

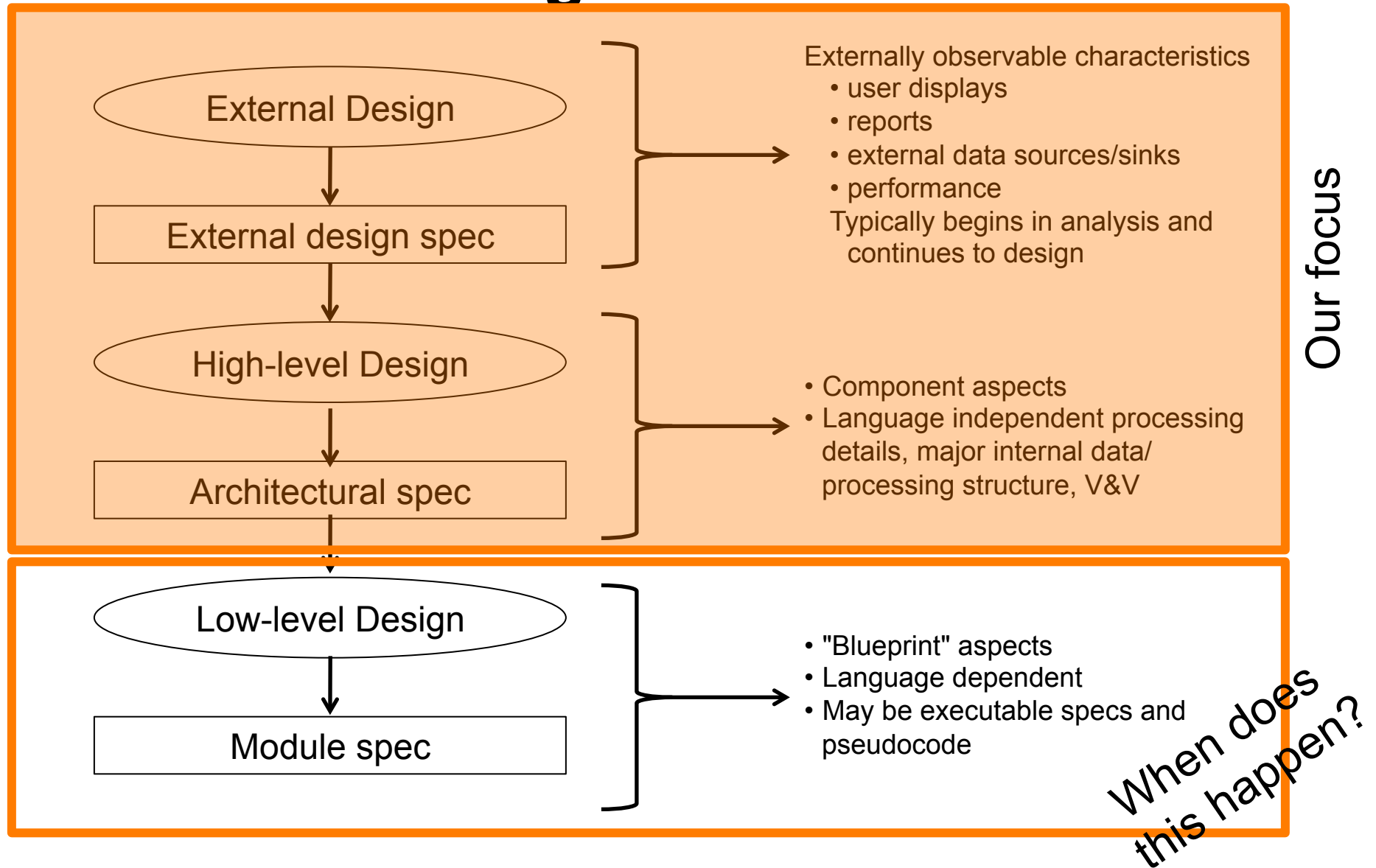
- estimators rarely think in detail at project beginning; thinking is done at component-level
    - we want to identify major components
      - where component = something used to represent a unit of work
        - » can be objects, functions, etc.

- Component characteristics

- should correlate closely to development costs
    - should be easy to visualize early in development
    - should be countable by automated means

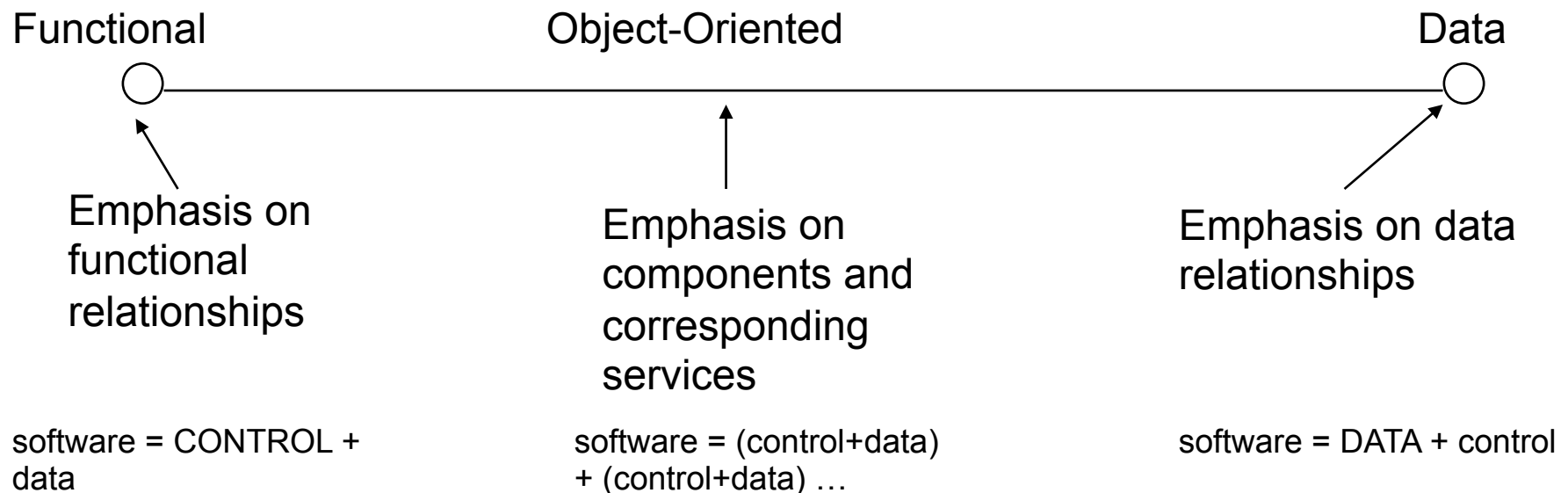


# Design activities



# Segue

- Component identification design:
  - construct rough model of solution: OO, FD, AO, etc.
  - follow rules of sound design (loose coupling, tight cohesion, etc.)
  - follow patterns, if so desired
  - determine granularity and nature of individual components
- Metaphysics: software = data + control
  - but, how much of each?



# An illustrative fairy tale

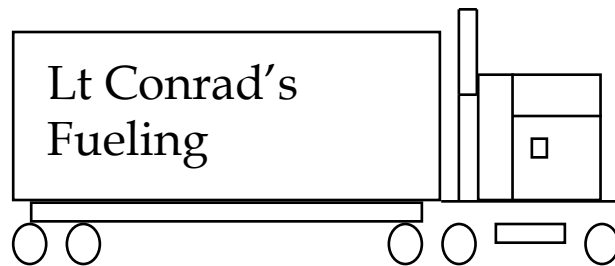
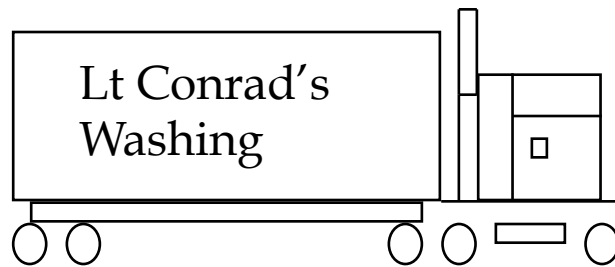
- Lt Conrad has been tasked to create a maintenance detachment. His mission:
  - Fuel fighters
  - Wash fighters
  - Fuel cargo planes
  - Wash cargo planes
- Lt Conrad must decide how to organize his outfit. Two alternatives:
  - Functional
  - Object-oriented

Lt Buff Conrad,  
Boy Wonder, Hero

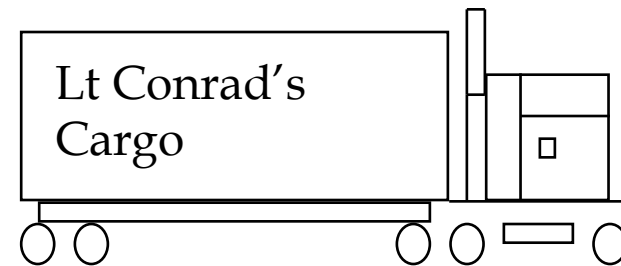
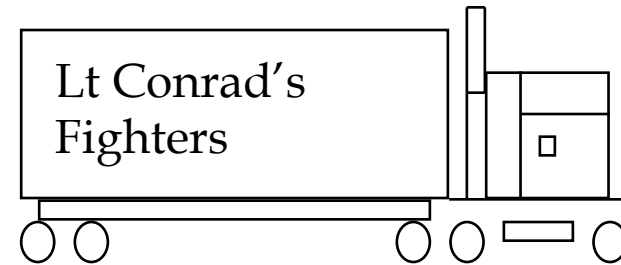


# Two alternatives: A picture

## Alternative 1: Functional

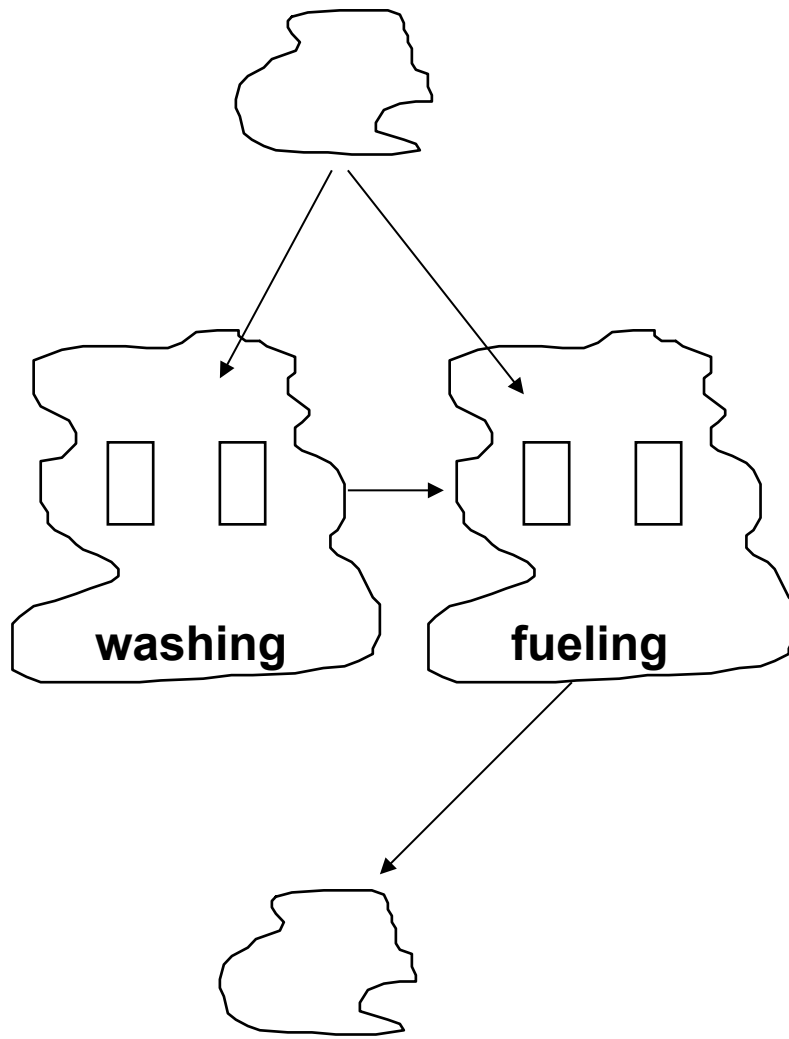


## Alternative 2: Object-Oriented

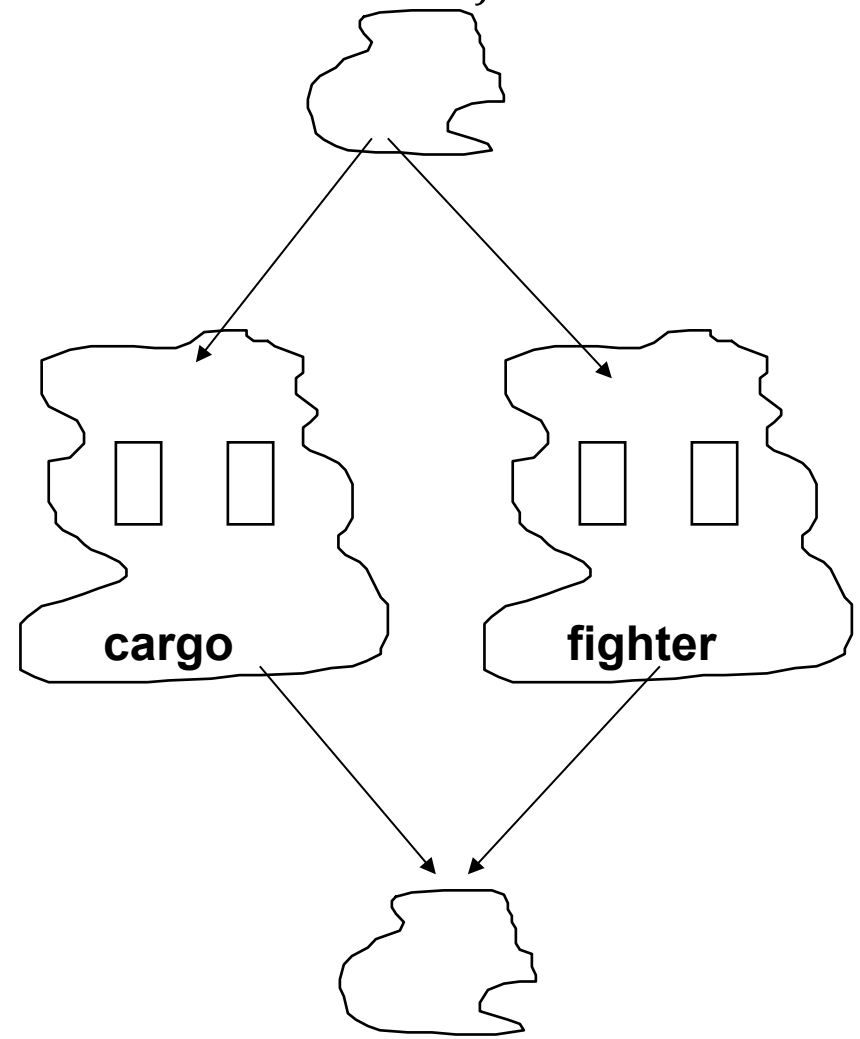


# Two alternatives: software perspective

Alternative 1: Functional



Alternative 2: Object-Oriented



# Assumptions

- Premise 1: we are interested in identifying components from which to calculate size, effort, time.
  - candidate components
    - class = component with zero or more methods
    - function = component with one method
- Premise 2: we may not be using a consistent – or pure – compositional philosophy
- Premise 3: components will map to code segments
- Premise 4: we will be building our conceptual design down to the level of “standard” components

# Conceptual Tools of Design

- Abstraction
  - management of detail
- Encapsulation
  - containment of detail
- Structure
  - composition of detail
- Behavior
  - operation of detail

# Patterns

- Concept:
  - proven design motif
- Classification
  - Creational patterns
  - Structural patterns
  - Behavioral patterns
  - Concurrency patterns



# Creational Patterns

Name	Description
Abstract factory	Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
Builder	Separate the construction of a complex object from its representation, allowing the same construction process to create various representations.
Factory method	Define an interface for creating a single object, but let subclasses decide which class to instantiate.
Lazy initialization	Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed.
Multiton	Ensure a class has only named instances, and provide global point of access to them.
Object pool	Avoid expensive acquisition and release of resources by recycling objects that are no longer in use.
Prototype	Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
Singleton	Ensure a class has only one instance, and provide a global point of access to it.

[http://en.wikipedia.org/wiki/Software\\_design\\_pattern](http://en.wikipedia.org/wiki/Software_design_pattern)

# Structural Patterns

Name	Description
Adapter	Convert the interface of a class into another interface clients expect.
Bridge	Decouple an abstraction from its implementation allowing the two to vary independently.
Composite	Compose objects into structures to represent part-whole hierarchies, allowing individual objects and compositions of objects to be treated uniformly.
Decorator	Attach additional responsibilities to an object dynamically keeping the same interface.
Facade	Provide a unified interface to a set of interfaces in a subsystem.
Flyweight	Use sharing to support large numbers of similar objects efficiently.
Front Controller	Provide a centralized entry point for handling requests.
Module	Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity.
Proxy	Provide a surrogate or placeholder for another object to control access to it.

[http://en.wikipedia.org/wiki/Software\\_design\\_pattern](http://en.wikipedia.org/wiki/Software_design_pattern)

# Behavioral Patterns

Name	Description
Blackboard	Generalized observer which allows multiple readers and writers to communicate information system-wide.
Chain of responsibility	Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request.
Command	Encapsulate a request as an object, allowing requests to be parameterized according to the clients
Interpreter	Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.
Iterator	Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
Mediator	Define an object that encapsulates how a set of objects interact.
Memento	Without violating encapsulation, capture and externalize an object's internal state allowing the object to be restored to this state later.
Null object	Avoid null references by providing a default object.

[http://en.wikipedia.org/wiki/Software\\_design\\_pattern](http://en.wikipedia.org/wiki/Software_design_pattern)

# Behavioral Patterns

Name	Description
Observer or Publish/Subscribe	Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically.
State	Allow an object to alter its behavior when its internal state changes.
Strategy	Define a family of algorithms, encapsulate each one, and make them interchangeable.
Template method	Define the skeleton of an algorithm in an operation, deferring some steps to subclasses.
Visitor	Represent an operation to be performed on the elements of an object structure.

[http://en.wikipedia.org/wiki/Software\\_design\\_pattern](http://en.wikipedia.org/wiki/Software_design_pattern)

# Concurrency Patterns

Name	Description
Active Object	Decouples method execution from method invocation that reside in their own thread of control.
Balking	Only execute an action on an object when the object is in a particular state.
Binding properties	Combining multiple observers to force properties in different objects to be synchronized or coordinated in some way.
Guarded suspension	Manages operations that require both a lock to be acquired and a precondition to be satisfied.
Join	Join-patterns provides a way to write concurrent, parallel and distributed programs by message passing.
Lock	One thread puts a "lock" on a resource, preventing other threads from accessing or modifying it.
Messaging design pattern	Allows the interchange of information (i.e. messages) between components and applications.
Monitor object	An object whose methods are subject to mutual exclusion
Reactor	A reactor object provides an asynchronous interface to resources that must be handled synchronously.

[http://en.wikipedia.org/wiki/Software\\_design\\_pattern](http://en.wikipedia.org/wiki/Software_design_pattern)

# Concurrency Patterns

Name	Description
Read-write lock	Allows concurrent read access to an object, but requires exclusive access for write operations.
Scheduler	Explicitly control when threads may execute single-threaded code.
Thread pool	A number of threads are created to perform a number of tasks, which are usually organized in a queue. Typically, there are many more tasks than threads. Can be considered a special case of the object pool pattern.
Thread-specific storage	Static or "global" memory local to a thread.

[http://en.wikipedia.org/wiki/Software\\_design\\_pattern](http://en.wikipedia.org/wiki/Software_design_pattern)

# Specification-Component Map

	Component 1	Component 2	Component 3		Component n
Scenario 1	operation 1a operation 1b	operation 2b			
Scenario 2		operation 2a operation 2b			
Scenario 3	operation 1a operation 1c		operation 3a operation 3b operation 3c		
Scenario m					operation na operation nb

# PCSE Architecture – Prong 2

- CRC cards
  - CRC = Class ... Responsibility ... Collaborator
  - think of them as a textual, tabular version of UML class diagrams

Component Name	
Design Approach	
Superclass	
Component Type	
Collaborators	
Operations	

oo  
or  
Functional

Logic  
Calculation  
Data  
Input/Output



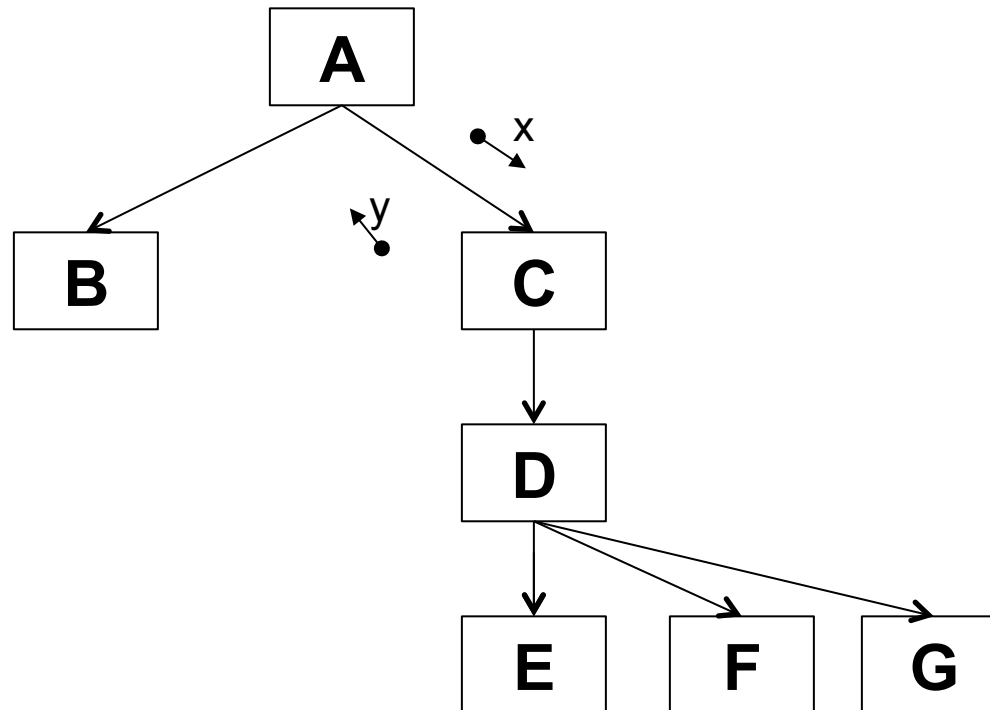
# PCSE Architecture – doin' it

- Identify a modeling philosophy
- Model the solution
  - repeat
    - identify components at a given level of abstraction
      - often this means starting with "glue" component
      - linguistics can aid decomposition:
        - » nouns and noun phrases => classes/objects
        - » verbs and verb phrases => methods or functions
    - identify component semantics
      - innate operations
      - service operations
    - identify relationships among components
      - collaboration is one-way
    - refactor
      - use patterns to refactor as appropriate
  - until each component ...
    - ... is associated with a standard type
    - ... exhibits sound design principles
      - coupling
      - cohesion

Component Name	
Design Approach	<input type="checkbox"/> Object-oriented <input type="checkbox"/> Functional
Superclass	
Component Type	<input type="checkbox"/> Logic <input type="checkbox"/> Calculation <input type="checkbox"/> Data <input type="checkbox"/> I/O
Collaborators	
Operations	

# PCSE Architecture – Prong 2

- Wire Diagrams
  - light-weight sketch of components and interactions among components



# PCSE Architecture

- Downstream
  - we want to extract architectural information from the code we produce
  - this represents "production output"
  - keeping a log of time gives us a way of estimating productivity ... and a way of forecasting effort

# OO Decomposition

```
class Notepad():
    def __init__(self):
        ...
    def addNote(self, note):
        ...
    def getNote(self, number):
        ...

class Command():
    def __init__(self):
        ...
    def listOfCommands(self):
        ...
    def setCommand(self, cmd):
        ...
```

... "main" code ...

what are the  
CRC cards?

what are the  
wire diagrams?

# Functional Decomposition

```
def clrScreen():
```

```
    ...
```

```
def makeSpace():
```

```
    ...
```

```
def printMenu():
```

```
    ...
```

```
def addNote(note):
```

```
    ...
```

```
def removeMessage(number):
```

```
    ...
```

```
... "main" code ...
```

# Hybrid Decomposition

```
class Note():
    def __init__(self):
        ...
    def addNote(note):
        ...
    def numMessages():
        ...
    def removeMessage(number):
        ...

def clrScreen():
    ...
def printMenu()"
    ...

... "main" code ...
```

# ? Decomposition

... "main" code ...

statement 1

statement 2

statement 3

...

statement n

**Kids: Do Not Do This At Home**

what are the  
CRC cards?



# Summary

## Topics

- Architecture defined
- Components
  - Architecture decisions
  - Requirements-Component map
  - Articulating component architecture
    - CRC cards
    - Wire diagrams

## Key Points

- Determining scope is the key objective of architecture
- Initial size/time/effort estimates depend on first-cut architecture of solution
- The "conceptual architecture" is an initial examination of components based on specifications
  - where typically 80% of design decisions are made
- We wish to articulate components in a fashion that provides us information needed. Light-weight examples:
  - CRC
  - Wire diagrams

Component Name	
Design Approach	<input type="checkbox"/> OO <input type="checkbox"/> FD
Superclass	
Component Type	<input type="checkbox"/> Logic <input type="checkbox"/> Calculation <input type="checkbox"/> Data <input type="checkbox"/> I/O
Collaborators	
Operations	

Component Name	
Design Approach	<input type="checkbox"/> OO <input type="checkbox"/> FD
Superclass	
Component Type	<input type="checkbox"/> Logic <input type="checkbox"/> Calculation <input type="checkbox"/> Data <input type="checkbox"/> I/O
Collaborators	
Operations	

Component Name	
Design Approach	<input type="checkbox"/> OO <input type="checkbox"/> FD
Superclass	
Component Type	<input type="checkbox"/> Logic <input type="checkbox"/> Calculation <input type="checkbox"/> Data <input type="checkbox"/> I/O
Collaborators	
Operations	

Component Name	
Design Approach	<input type="checkbox"/> OO <input type="checkbox"/> FD
Superclass	
Component Type	<input type="checkbox"/> Logic <input type="checkbox"/> Calculation <input type="checkbox"/> Data <input type="checkbox"/> I/O
Collaborators	
Operations	

Component Name	
Design Approach	<input type="checkbox"/> OO <input type="checkbox"/> FD
Superclass	
Component Type	<input type="checkbox"/> Logic <input type="checkbox"/> Calculation <input type="checkbox"/> Data <input type="checkbox"/> I/O
Collaborators	
Operations	

Component Name	
Design Approach	<input type="checkbox"/> OO <input type="checkbox"/> FD
Superclass	
Component Type	<input type="checkbox"/> Logic <input type="checkbox"/> Calculation <input type="checkbox"/> Data <input type="checkbox"/> I/O
Collaborators	
Operations	