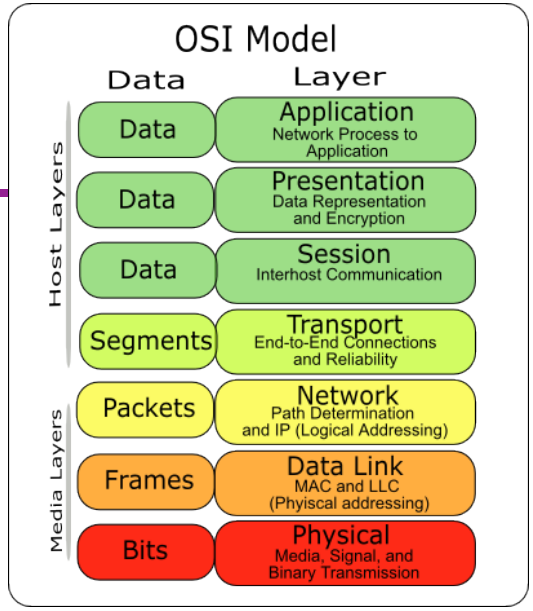
CSE403 ● Software engineering ● sp12

Week 4				
Monday	Tuesday	Wednesday	Thursday	Friday
• Composition • Reading III due	• Group meetings	• Phil Kimmey on using git @ rover.com	• SDS++ due	• Midterm review – content, format • Progress report due

Today

- Orthogonality in design: when, if ever, can we use multiple good design ideas simultaneously?
- Ex: generic collections in Java
  - Abstraction over the details of the collection (array, list, hashtable, etc.)
  - Separate abstraction over the values – that is, the type of the elements
- This kind of orthogonality is very powerful

Layering in one slide



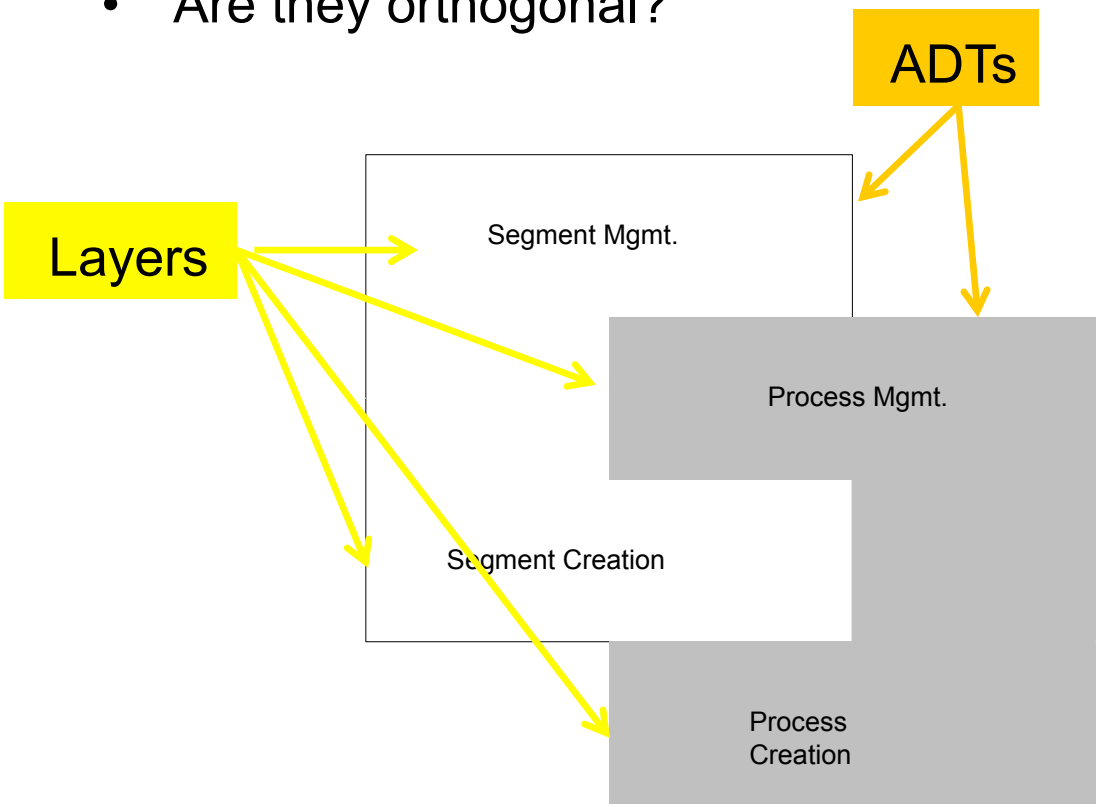
- Used in part for program families, systems that have "so much in common that it pays to study their common aspects before looking at the aspects that differentiate them" [Parnas 1979]
  - For example, Microsoft operating systems, a number of the Mozilla systems, …
- Another kind of dependence useful for families
  - A module **`A`** **`uses`** a module **`B`** if the correctness of **`A`** depends on the presence of a correct version of **`B`**
- A non-hierarchical **`uses`** relation makes it difficult to produce useful subsets of a system

- **`uses`** and **`invokes`** dependences often but do not always coincide
- Invocation without use: name service with cached hints
- Use without invocation: examples?

```
ipAddr := cache(hostName);
if wrong(ipAddr,hostName)
  ipAddr := lookup(hostName)
endif
```

ADTs and layering interact?

- Information hiding modules (say, ADTs in this case) and layers are distinct concepts
- How and where do they overlap in a system?
- Are they orthogonal?

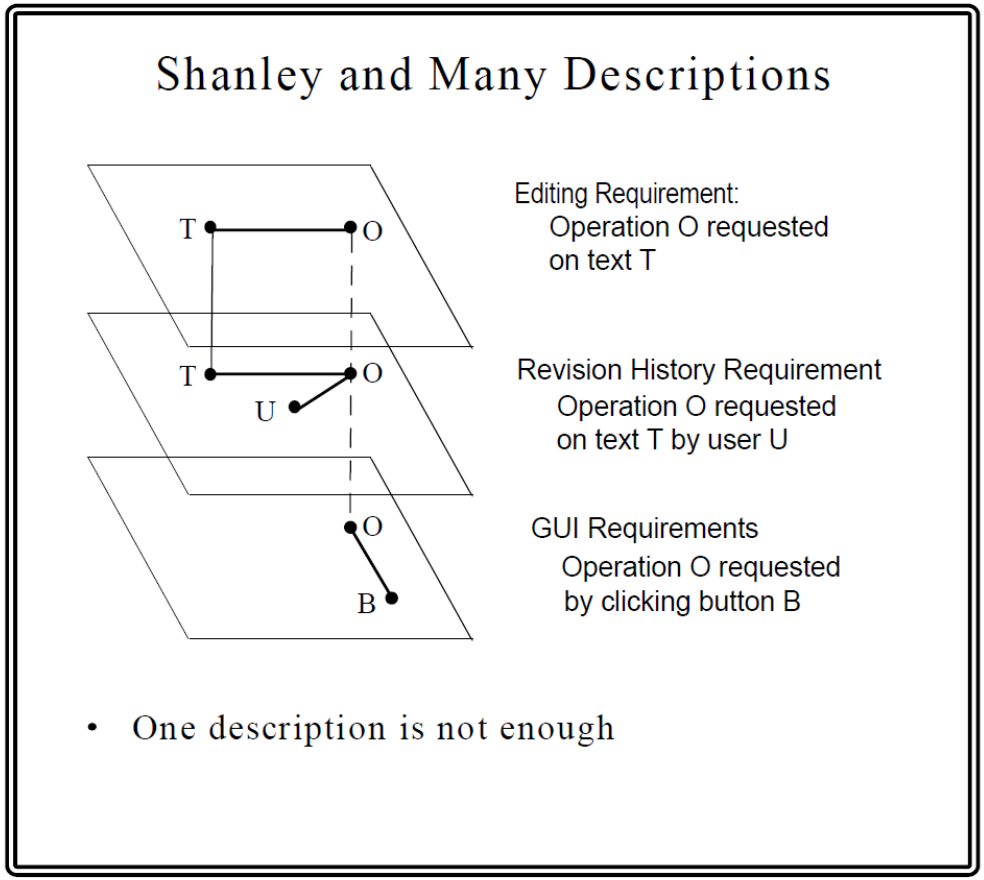


What kinds of secrets (potential changes)? What kinds of families?

Extra credit

Composition: Michael Jackson

- Jackson observes that we sometimes overvalue the notion of hierarchical decomposition
  - The world itself does not have strict typing and inheritance hierarchies
- He argues that the CMYK printing is a better analogy for software composition at many levels



Design patterns

- What are they?
- Why are they?

Following slides from 331 au11
It'll probably be a whirlwind

What is a design pattern?

- A standard solution to a common programming problem
  - a design or implementation structure that achieves a particular purpose
  - a high-level programming idiom
- A technique for making code more flexible
  - reduce coupling among program components
- Shorthand for describing program design
  - a description of connections among program components (static structure)
  - the shape of a heap snapshot or object model (dynamic structure)

Why design patterns?

- Advanced programming languages like Java provide lots of powerful constructs – subtyping, interfaces, rich types and libraries, etc.
- By the nature of programming languages, they can't make everything easy to solve
- To the first order, design patterns are intended to overcome common problems that arise in even advanced object-oriented programming languages
- They increase your vocabulary and your intellectual toolset

From a colleague

- FML. Today I got to write (in Java):

```
import java.util.Set;
import com.google.common.base.Function;
import com.google.common.collect.DiscreteDomains;
import com.google.common.collect.Iterables;
import com.google.common.collect.Ranges;

final int x = ...;
Set<Integer> indices =
   Ranges.closed(0, size).asSet(DiscreteDomains.integers());
Iterable<Coord> coords =
  Iterables.transform(indices, new Function<Integer,Coord>(){
    public Coord apply (Integer y) {
      return new Coord(x, y);
    }
  }
);
```

when I wanted to write (in Scala):

```
val x = ...;
val coords = 0 to size map(Coord(x, _))
```

No programming language is, or ever will be, perfect.

Extra-language solutions (tools, design patterns, etc.) are needed as well.

Perlis: "When someone says 'I want a programming language in which I need only say what I wish done,' give him a lollipop."

Whence design patterns?

- The Gang of Four (GoF) ⓘ – Gamma, Helm, Johnson, Vlissides
- Each an aggressive and thoughtful programmer
- Empiricists, not theoreticians
- Found they shared a number of "tricks" and decided to codify them – a key rule was that nothing could become a pattern unless they could identify at least three real examples

My first experience with patterns at Dagstuhl ⓘ with Helms and Vlissides

Patterns vs. patterns

- The phrase "pattern" has been wildly overused since the GoF patterns have been introduced
- "pattern" has become a synonym for "[somebody says] **X** is a good way to write programs."
  - And "anti-pattern" has become a synonym for "[somebody says] **Y** is a bad way to write programs."
- A graduate student recently studied so-called "security patterns" and found that very few of them were really GoF-style patterns
- GoF-style patterns have richness, history, language-independence, documentation and thus (most likely) far more staying power

An example of a GoF pattern

- Given a class C, what if you want to guarantee that there is precisely one instance of C in your program? And you want that instance globally available?
- First, why might you want this?
- Second, how might you achieve this?

Possible reasons for Singleton

- One **`RandomNumber`** generator
- One **`Restaurant`**, one **`ShoppingCart`**
- One **`KeyboardReader`**, etc…
- Make it easier to ensure some key invariants
- Make it easier to control when that single instance is created – can be important for large objects
- …

Several solutions

```
public class Singleton {
  private static final Singleton instance
      = new Singleton(); // Private constructor prevents
                         // instantiation from other classes
  private Singleton() { }
  public static Singleton getInstance() {
    return instance;
  }
}
```

Eager allocation of instance

```
public class Singleton {
  private static Singleton _instance;
  private Singleton() { }
  public static synchronized Singleton getInstance() {
    if (null == _instance) {
      _instance = new Singleton();
    } return _instance;
  }
}
```

Lazy allocation of instance

And there are more (in EJ, for instance)

Abstract Factory Pattern

- wikipedia
- stackoverflow
- …and who knows where?!

Points to make

- Lots of different kinds of dependences
- Composition of different design/abstraction mechanisms can be extraordinarily powerful
  - So increasing your understanding of powerful mechanisms and patterns will surely help simplify your design over time
  - This in turn may well increase some form of conceptual integrity

CSE403 ● Software engineering ● sp12

Week 3-4				
Monday	Tuesday	Wednesday	Thursday	Friday
• Composition • Reading III due	• Group meetings	• Phil Kimmey on using git @ rover.com	• SDS++ due	• Midterm review – content, format • Progress report due