Software frameworks/cadavers

- Experience with OO programming and design shows that design patterns are useful
  - Where do we get the experience?
  - How do we impart experience?
  - What can we use to illustrate patterns in practice?
  - What patterns should we emphasize, how, when?

*Good design comes from experience, experience comes from bad design*

Fred Brooks/Henry Petroski

- Good design also comes from experiences with good design
  - use software cadavers to show patterns in practice
Factory Methods/Factory Pattern

- Suppose we’re using a Printer hierarchy in Proofread (e.g., as in `classwlines.cpp` with normal printer, print-in-order-by-occurrences printer, …). Where does the Printer object being used come from?
  - The real program needs a Printer object, but doesn’t know what kind it needs, just when it’s needed
  - Separate creation/construction from use, localize knowledge of which (or how) a Printer is created

- Create a factory method aka virtual constructor
  - We can change the factory code, or create new factories

```cpp
Printer * p = factory.makePrinter();
// use p here, don’t know what kind we have
```
Factories continued

- **Suppose you want to experiment/use different implementations as part of Proofread**
  - Use hashmap, treemap, trie, super-duper-wingdoodle, ...
  - The interfaces for classes doesn’t need to change, e.g., consider TreeMap and HashMap in Java, both implement Map
  - We may have more than the Map-like class, there may be associated Printers, Contextors, ...

- **Abstract Factory defines how to create products used by clients: Map, Printer, Contextor, ...**
  - Clients create a concrete factory and use it, hashfactory, triefactory, ...
  - Consider Motif, Mac, Windows, ..., LAF for GUIs
Simple Hangman factory

- **We want to use a Word without knowing where the word comes from**
  - Hard-wired, from a file, from a URL, program resource, ...
  - The source of the word is variable, not the Word
    - This is different from previous uses of factory, how?
    - Could implementation of word change based on source?

- **Create a WordFactory abstraction/interface, instantiate it with concrete Word sources**
  - DefaultWordFactory in hangman example, from files?
  - Where is Factory instantiated, how is factory tested?
  - Do we want more than one Factory in a program?
Singleton Pattern

- How can we control object creation, e.g., limit to one instance
  - Common example: modal dialog box
    - User must respond, how many are needed in a program?
  - One instantiation, clients should know how to get it
    - What about global variable?
  - Want to allow subclassing
    - How does a class with static methods compare?

- Typical solution
  - Protected constructor, static pointer to real instance, factory-like method for getting instance
  - See singleton.cpp and Singleton.java

- Consequences
  - What about delete and garbage collection? Troublesom
Observer/Observable, aka MVC

- **We want to play hangman, but change how the gallows appears**
  - In text version we could show just __ __ __ __
  - We could use ASCII art to draw stick figures
  - We could use a GUI to draw more realistic figures
  - The *View* should be separate from the *Model*
    - Model is number of misses, max # misses (for example)
    - View is how model appears to user, could be multiple views
  - The relationship between the model and the view may be handled by a third-party, the *Controller*
    - Mediates between the two classes Model and View

- **Model, View, Controller is called Observer/Observable in GOF**
  - Observer terminology directly supported in Java (but ugly?)