Pattern Language

A long time ago ... in 1977.

Christopher Alexander published, "A Pattern Language".

His concept was to create a set of terms that could describe common patterns in architecture.

“alcoves” “entrance transition”
“wings of light” “a place to wait”
“flexible office space”
Pattern Language

- This pattern language would allow designers to talk about architectural concepts **abstractly**.

- It also provided a foundation for promoting **commonality** of design.

“Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to the problem, in such a way that you can use this solution a million times over, without doing it the same way twice.”

- Christopher Alexander

---

Pattern Languages in Software

Ten years later in 1987...

Kent Beck and Ward Cunningham propose using a pattern language in programming in their paper at OOPSLA, “Using Pattern Languages for Object-Oriented Programs”.

http://c2.com/doc/oopsla87.html

These **patterns** revolved around the user interface:

“Window Per Task” “Few Panes Per Window”
“Standard Panes” “Short Menus” “Nouns and Verbs”
Design Patterns

Then in 1995, the Gang of Four (GoF) published the foundational book, “Design Patterns”.

The identified 23 common design patterns for developing OO software.

Design Patterns

- A design pattern is an abstract solution for common software problems.

- These designs are general approaches that can be applied in various situations.

- Because they are abstractions to problems they allow developers to talk about design without worrying about implementation details.
Design Patterns

- Design patterns are not specific to a language, but are used with OO languages, because they center around objects.

- They promote the reuse of code and commonality in design so that we can have more meaningful communication.

You may ask, “How should I implement my user interface for my standard deck?”

And I can respond, “Have you considered using the observer pattern?”

How to describe a design pattern...

- There are some common parts to describing a pattern:
  - Name – A short description to identify what the pattern is.
  - Problem – What problem is the pattern attempting to solve? In what situations is this design pattern applicable?
  - Solution – A description of the solution outlining the responsibilities and collaborations. Not an implementation, but an abstract design. May include UML designs.
  - Consequences – Why is this pattern useful? Are there any trade-offs to using this design pattern?
A real world example...

■ I have the original Star Wars movies on VHS.
  
  – *The ones before George Lucas kept changing everything.*

  – *The one where Han shoots first.*

Source: http://swhomevid.site40.net/9092box.html

A real world example...

■ I have an old VCR to play them on.

Source: http://en.wikipedia.org/wiki/Videocassette_recorder
A real world example...

- I have a brand new HD TV to view them.

Source: http://www.definingsystems.com/audiovideo/hdtv-wall-mounting/

A real world example...

- The problem is, my VCR outputs are RCA:

- and my TV’s input is HDMI:
A real world example...

- What can I use to watch my VHS tape on my HD TV?


The Adapter design pattern

- Problem:
  - I need to convert the interface of a class into another interface the client expects. The adapter pattern lets classes work together that couldn’t otherwise because of incompatible interfaces.

  - The adapter pattern can be used in a situation where we are attempting to use classes together that have different interfaces.
The Adapter design pattern:

Solution:

* Note I could use inheritance or composition.
The Adapter design pattern

- Consequences:
  - The adapter pattern basically provides a class wrapper.
  - You create a new class that wraps the functionality of an existing class with a new interface.
  - Allows me to reuse an existing class.