Stage 1: video 2

Design patterns are *conceptual* tools for solving complex software problems. We'll be discussing ones that appear in Cocoa, but they exist in all programming languages and some span multiple languages. For example, MVC is applicable to any application driving a user interface, regardless of the language it's written in.

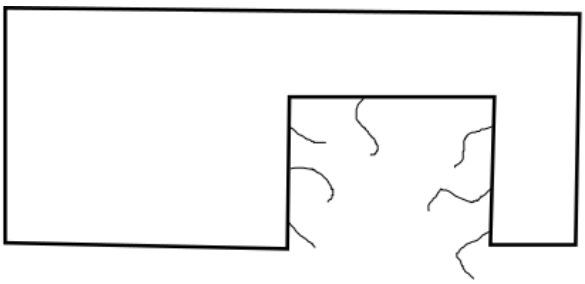
Design patterns appear at different levels of application structure from the organization of code modules, to data structure creation, to communication. Some like *two-stage object creation*, you are already using, while others, like *key-value-observing* (or KVO) are more esoteric and not found in every project.

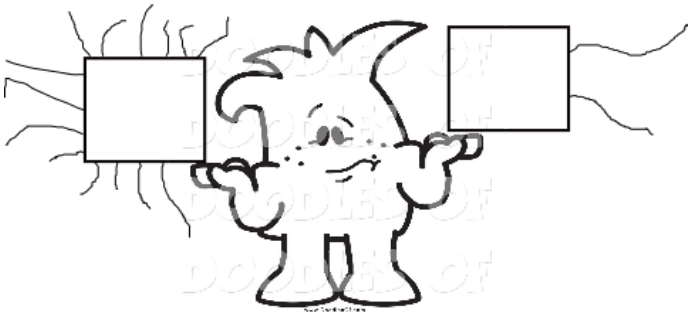
These patterns are simple and elegant solutions that have evolved over time and many have become generally accepted as the "best way" to address certain design challenges.
They are not, however, "ready made fixes" that can be cut and pasted into your application. They are more like suits that need to be tailored to your specific needs.

Design patterns are the *core* of solutions to common software design problems. Most simply, they are well-thought-out and well-tested "ways of doing things" that benefit software at every stage of it's lifecycle from development, to maintenance, through eventual replacement.

They solve problems by promoting common software *virtues*. For example, a number of the patterns foster *decoupling*. Decoupling is nearly always a good idea in software; objects that are loosely-coupled with others may be more easily replaced, modified and/or reused in other projects.

<Part with wires graphic>





Coupling is analogous to the number of wires connecting a part in an electrical system. Removing or reconnecting a part with 2 wires is much less trouble than doing so with a part that has 20. Not only does splicing 20 connections take more time, but it requires knowing the function of all 20. More time, effort and complexity makes it more likely there will be an error in connecting the part.

All things being equal, swapping out and/or reusing the part with 2 wires is a much easier job.
</part with wires graphic>

Simplifying channels of communication between objects is one way to promote decoupling.
A number of the patterns we'll look at: *KVO*, *Notifications* and (to some extent) *Singletons* foster decoupling by simplifying message passing between objects and limiting the amount of information needed for them to start "talking".

Asside from making you a better programmer, perhaps the best reason for an iOS developer to learn Cocoa design patterns is that they are part of the vocabulary of software development. Not only are they referenced in Apple's documentation, but they are a common topic in discussions between developers.

Names of design patterns are not simply software jargon, but a way of communicating complex and nuanced approaches to a software design.

In the next video, we'll take a brief tour through the design patterns we'll be looking at.

Questions:

1. The built-in XCode debugger is:

- LLVM
- GDB
- LLDB

(ans. LLDB)

2. True or False: Design patterns only exist in Cocoa. (ans. false)
3. True or False: All the design patterns you know should always be applied to all your software development projects. (ans. false)

4. What does KVO stand for? (ans. key-value-observing)
5. What does MVC stand for? (ans. model-view-controller)