Stage 1: video 1

Scene 1: On-Set

Hello, my name is Bjorn and welcome to Design Patterns in Cocoa. I have over 20 years of programming experience in both industry and academia. In addition to iOS, I've taught Computer Architecture, Assembly Language, C, and Robotics.

This course is intended to give intermediate-level iOS programmers a deeper understanding of some common design patterns that appear in the Cocoa development frameworks. Understanding the "*Hows?*" and "*Whys?*" of these patterns will increase your productivity and enhance the quality of your code. Working *with* the patterns, rather than mistakenly working *against* them is essential for gaining greater proficiency as an iOS developer.

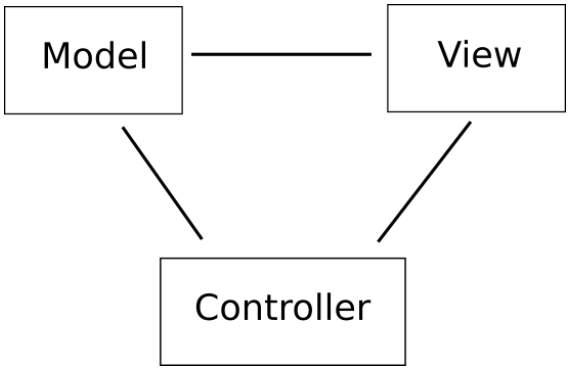
In this series of videos, we'll be examining five common Cocoa patterns. You've likely seen these before, but may not have recognized them as design patterns or fully understood the reasoning behind their use. This course will teach you to both recognize design patterns, and successfully apply them in your own code.

<clip of LLDB activity>

For some patterns, we'll present simple examples and demonstrate their operation using some of the advanced debugging capabilities of the XCode environment.
The XCode debugger, LLDB, will allow us to follow program execution and see exactly when and how critical data values change. Hopefully, in the process of learning the mechanics of how these examples operate, you'll also pick up a few useful debugging tricks.

</clip of LLDB activity>

<MOTION: show source code modules moving into the Model, View, and Controller boxes>



We'll look at other, higher-level patterns, like Model-View-Controller (or MVC), in terms of how they can be used to structure your application.

MVC defines functional subsystems for displaying data(the View), managing data(the data Model) and mediating between the two (the Controller).
Organizing your source-code modularly in accordance with the MVC design pattern will structure your development process, reduce the likelihood of bugs and maximize opportunities for code reuse.
</MOTION>

For each design pattern, we'll discuss the reasoning behind it: the problem it solves, and why it's a good approach as compared to the alternatives.

We'll also discuss *best practices* for each pattern: when and how they should be applied and how to avoid *misapplication*.

Design patterns are tools, and as such, each one is appropriate for certain situations, but not for others. They can be misapplied or overused so it is important to consider the implications of using a given pattern.

<add graphic showing a wrench and a hammer and demonstrating the use and misuse of each, e.g. hammering a nail with the wrench, hammering on a nut, etc...>

Taking an analogy from the mechanical world: imagine that you're disassembling some appliance and find yourself facing a nut on a threaded bolt. You know that the job will call for a wrench. You're not yet sure what kind of wrench (adjustable, socket, monkey…) since you haven't yet examined the nut shape, its access and the clearance...., but you can be fairly certain that some sort of wrench will be needed.

A given design pattern is something like the "wrench" in this analogy. Familiarity with a "toolbox" of patterns will allow you to recognize when a certain pattern is applicable: when to use a "wrench" and when to use a "hammer". You may not immediately know all the implementation details, but you'll know which pattern can be tailored to solve the problem.

.**<end tool graphic>** Move <end tool graphic> to here, and cut to on-camera for the explanation of the analogy.

Learning if and how to apply these patterns is at least as important as knowing the patterns themselves.

When considering when to apply a design pattern, it's a good idea to ask yourself the follow questions:

<Keynote showing questions>

- Does using the pattern reduce the overall complexity of the application?
- If the size and/or scope of the project grows, will this use of the design pattern scale well along with it?
- Is this the correct pattern to be using? Is there another that might be more appropriate in this case?
- Will this use of the pattern contribute to the modularity and maintainability of the codebase?
- Are there any downsides to applying the pattern? E.g.: Will it hinder unit testing? Might it hurt performance?

</Keynote>

By the end of this course you should understand design patterns well enough that you'll have a "feel" for when a pattern is a good fit for a given development problem.

Okay, let's start learning about design patterns...