E25 - DESIGN PATTERNS

Written by

Robert C. Martin
(Uncle Bob)

Segment 1 - Welcome

1 FRONT DOOR 1

UNCLE BOB
Welcome, welcome, to Clean Code episode 25, Design Patterns. Here, let me take your hat.

Now I know what you're thinking. You're thinking that I'm leaking out on my promise to do a case study of TDD, including GUI and DB. Right?

Well, I'm not. It's just that I started thinking about trying to fit a reasonable case study into an hour long video and realized that there simply wasn't any way to do that well.

And then it occurred to me. What better vehicle for Design Patterns could there be, than a significant case study that spans multiple episodes?

So that's what we're going to do. We're going to start a new series that will run in parallel with this one. In that series we will build an application using everything we've learned so far; and everything we are about to learn.

2 OFFICE 2

UNCLE BOB
So, in this episode we're going to begin our journey through the topic of Design Patterns. And it's not a moment too soon, because we're going to need those patterns for the application we'll be developing in the case study series.

First we'll look at what design patterns are; and we'll study a few examples from outside the software domain.

(MORE)

UNCLE BOB (CONT'D)
We'll introduce the Design Patterns book and take a quick walk through it's pages.

Then we'll focus on just one of the patterns -- the Command Pattern. We'll show you it's structure, and many of it's common uses.

We'll see how you can use the Command Pattern to decouple what is done from who does it and when it is done.

We'll also see that the Command Pattern provides an interesting solution to implementing Undo behavior.

Finally we'll look at the actor model of concurrency, and see how the Command Pattern helps us implement that.

3 MINECRAFT 3

Note: VO

Blowing up TNT

ENGINEER
So head for them thar hills, pardners, 'cuz we're about to blast our way into the world of design patterns.

Segment 2 - Neutron Stars

4 WORKSTATION 4

UNCLE BOB
Back in Episode 19 we talked about how massive stars end their lives when they've burned through the last of their fuel producing core of iron. Unable to produce the engery necessary to withstand it's own gravity, the iron core collapses so tightly that the protons and electrons react to form neutrons, which arrange themselves into a sphere at the center of the star. The energy liberated by this collapse blows the star to smithereens as a type II supernova.

5 KITCHEN 5

UNCLE BOB
Now that explosion is pretty spectacular; and it's also the forge in which most of the atoms in our everyday world are created. But what about that ball of neutrons? What happens to it?

6 OFFICE 6

UNCLE BOB
Sometimes the explosion is so forceful that the ball of neutrons doesn't survive. We'll talk about that in another episode. Often, however, the ball of neutron rides through the storm of the explosion, and is left behind as a stable object. We call it a neutron star.

7 1737 TALL PINE WAY 7

UNCLE BOB
Now remember, neutrons are fermions. They obey the Pauli Exclusion Principle.
(MORE)

UNCLE BOB (CONT'D)
Two neutrons cannot be in the same system and also be in the same state. So, the neutrons in that ball must arrange themselves into an energy shell structure analogous to the electrons in an atom.

8 COUCH 8

UNCLE BOB
The more neutrons there are, the more energy levels there must be. The higher the energy of the neutrons, the more momentum those neutrons carry. That momentum create a pressure that pushes back against the neutron star's gravity. Remember that force is called degeneracy pressure.

9 WHITEBOARD 9

Note: D Density = 10^15, Surface Gravity 10^11 g.

UNCLE BOB
And what a pressure that is! A typical neutron star has twice the mass of our Sun, but is only about six miles in diameter. It has the density of an atomic nucleus, or about a quadrillion times that of water. That combination of mass and density creates a gravitational field intense enough to make a pint of water weigh 100 billion pounds.

So this is a pretty interesting object. But it gets better.

10 RECROOM 10

Note: V spinning skater.

UNCLE BOB
Stars in general spin. Prior to it's collapse, the parent of the neutron star was spinning too.
(MORE)

UNCLE BOB (CONT'D)
But then the core collapsed by a factor of 10,000 or so, which, like a skater pulling in her arms, caused the neutron star to spin up to a rate of, say 60 rotations per second.

11 CAR 11

UNCLE BOB
So now we've got an object that's six miles in diameter, and twice the mass of the sun, spinning at 3600 RPM, the speed of a standard hard disk. Do you think there's some kinetic energy in this thing?

12 SCREEN PORCH 12

UNCLE BOB
Oh, but it gets better. The parent star had a big magnetic field. When the core collapsed, that field got compressed by a factor of 10,000 or so. So the field intensities at the surface of the neutron star are perhaps a trillion times the Earth's magnetic field. That's strong enough to rip most materials apart.

13 WORKSTATION 13

UNCLE BOB
Now let's think about this for a minute. Here's an object, with a truly radical magnetic field, spinning at 3600 RPM. What do you get when you spin a magnetic field? That's right, you get an electric field. You get a voltage. In this case a whopping big voltage!

14 WHITEBOARD 14

Note: D pulsar

UNCLE BOB
That voltage aligns itself with the poles of the magnetic field and accelerates charged particles like electrons that happen to be within it's range. But don't forget about that magnetic field. As electrons move through a magnetic field they rotate around the field lines. And what do rotating electrons do? They give off photons. If the field strenghts are high enough, those photons will have frequencies ranging from radio waves to visoble light. And all those photons blast outwards of the poles of the neutron star in two intense beams.

15 GS 15

Note: J Mars

Note: D neutron star lighthouse.

UNCLE BOB
Now it just so happens that the magnetic poles of the neutron star don't often align with the rotational poles of the star. So these two beams of light swing around the sky like some kind of crazy lighthouse. If the Earth happens to be in the path of those beams, then we see that lighthouse flashing at us.

We call such flashing objects Pulsating Stars, or Pulsars.

16 SUN ROOM 16

UNCLE BOB
Now it takes a lot of energy to spin a stellar magnetic field and radiate those beams of light. That energy comes from the kinetic energy of the neutron star's rotation. And so, with time, the neutron star slows down.
(MORE)

UNCLE BOB (CONT'D)
As it slows, the electric field shrinks, the electrons slow their rotations, and the beams gradually decrease their energy from visible light down toward the radio part of the spectrum.

17 FRONT PORCH 17

UNCLE BOB
But we can still see them, flashing a way once or twice per second in the radio spectrum. Indeed, the very first pulsar to be found, was a radio pulsar. And, in fact, almost all the pulsars we know of are radio pulsars. Visible pulsars are short lived because they burn through their kinetic energy pretty quickly.

18 GS 18

Note: J Crab Nebula

UNCLE BOB
Still, there is one very famous visible pulsar in the Crab Nebula, 6000 light year away in the constellation of Taurus.

The crab is a remnant of a type II supernova that blew up on July 4, 1054 AD. It was reported by the Chineese who said it was visible in the night time for two years, and could be seen in the daytime for the first month.

When we look, now, at the very center of the Crab, we see a strong radio pulsar spinning 30 times per second. We also see dim flashes of visible light that are synchronized with the radio pulses.

19 WORKSTATION 19

UNCLE BOB
Overall, we know of several thousand pulsars scattered around teh sky. Some spin at leisurely rates of a few RPM, and others spin crazily a thousand times per second. The regularity of the pulses is so consistent, that nowadays we use them to calibrate our atomic clocks.

Segment 3 - Design Patterns

20 WORKSTATION 20

UNCLE BOB
So, before we begin looking at the requirements of our case study, it would be a good idea if we took a moment to talk about design patterns. Can anyone tell me what they are?

21 GS 21

RUBY ROD
Well, like, y'know, about a million years ago these four dudes wrote this book named Design Patterns; and it was all about patterns of design.

22 WORKSTATION 22

Note: P Design Patterns Book.

Note: 2

UNCLE BOB
(Sarcastic)
Thank you for that, Ruby, that was very helpful. Yes, here's that book. It was written in 1995 by Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm -- affectionately known as the Gang of Four.

This book is probably the most important book written about software in the last thirty years. Not because it contains anything new; but for precisely the opposite reason. What this book talks about is old ideas. Ideas that have withstood the test of time.

So, just what is a Design Pattern

GS

DANNY DOTNET
Oh, I know, Uncle Bob, I know. A design pattern is a solution to a problem in a context.

23 WORKSTATION 23

UNCLE BOB
A competent answer, Danny -- virtually a quote from the text. But what does that mean?

24 GS 24

JERRY JAVA
Yeah, yeah, it means you can solve problems with patterns.

25 WORKSTATION 25

UNCLE BOB
Well, sort of -- and sort of not. But perhaps an example would be useful.

Title: Bisected Oval

26 WHITEBOARD 26

UNCLE BOB
Have you ever looked through one of those “How to Draw” books that show you the steps to draw a face? Often they start with an oval, like this. Then they bisect that oval vertically and horizontally. Then they bisect the lower vertical here, and again here.

Now you can draw the eyes, nose, mouth, ears, and hair.

This technique is a pattern. We might call it the Bisected Oval pattern. As Danny said, it is a solution to a problem in a context.

The problem is to draw a face. The context is a line drawing, facing front.

27 COUCH 27

Note: P Green Card

UNCLE BOB
If your problem is to draw a face, and you don't mind a line drawing facing front, then you can use the Bisected Oval pattern.

28 KITCHEN 28

UNCLE BOB
Notice that I gave the pattern a name. That's a key element to any pattern. Danny said that a pattern is a solution to a problem in a context. I'll amend that definition to: A named solution to a problem in a context.

29 GS 29

RUBY ROD
Man, like, what's so great about the name? I mean what counts is what you do, not what you call it.

30 WORKSTATION 30

UNCLE BOB
The importance of the name is that we can talk about the pattern. We can use the name to communicate our intended solution.

31 GS 31

Note: J Washington_Crossing_the_Delaware.jpg

UNCLE BOB
Let's say that we're artists who are supposed to create a oil painting mural of George Washington crossing the Delaware river. We can talk about how we're going to solve that problem by using the names of the patterns. We can ask: "Shall we use Bisected Oval"

32 GS 32

Note: J Washington_Crossing_the_Delaware.jpg

RUBY ROD
Woah, man, you can't use that pattern because, like, George isn't facing front -- he's in profile.

33 GS 33

Note: J Washington_Crossing_the_Delaware.jpg

JERRY JAVA
Yeah, yeah, and it's an oil painting, not a line drawing.

34 WORKSTATION 34

Note: A Bisected Oval gets crossed out on screen.

UNCLE BOB
Right. The context for the pattern is incompatible with the task. So the Bisected Oval pattern can't be used for Washington's face.

35 FAMILY ROOM 35

UNCLE BOB
That's the value of a named pattern. If you know the patterns and their names then as your team discusses design strategy, you can refer to the patterns by name, and quickly decide whether the context is appropriate.

36 GS 36

DANNY DOTNET
OK, Uncle Bob, I get that. But why should I learn these patterns. I mean, I already know how to program.

37 OFFICE 37

Note: P Chessboard

UNCLE BOB
Do you play Chess, Danny?

38 GS 38

DANNY DOTNET
Sure, all the time. Wanna play?

39 OFFICE 39

Note: P Chessboard

UNCLE BOB
Perhaps later. Have you read any books on chess?

40 GS 40

DANNY DOTNET
Well, yes, actually, I have. I read a book by Bobby Fisher once. It taught me a lot.

41 OFFICE 41

Note: P Chessboard

UNCLE BOB
Do you know how many books have been written about chess. I mean have you ever walked through the chess section of a library or a book store and seen the huge number of books there?

42 GS 42

DANNY DOTNET
Yes, Uncle Bob, I have. There are so many. There are books about the opening, books about the endgame, books about the middle game, books for beginners, books for masters, there are a lot of books about chess.

43 OFFICE 43

Note: P Chessboard

UNCLE BOB
Right you are. There are thousands of them. And that's a measure of how much there is to know about the game.

But consider the first time you played a game of chess. You had just learned a few simple rules. You knew none of the strategy. You hadn't read any of the books. You just started moving the pieces in accordance to the rules.

44 GS 44

DANNY DOTNET
Yeah, Uncle Bob. I wasn't a very good chess player back then.

45 OFFICE 45

Note: P chessboard

UNCLE BOB
Nobody is Danny. When you first learn the rules, you don't know any of the principles. You don't understand the value of the center four squares. You don't know the relative value of the pieces. You don't understand the power and threat of a lone pawn.

46 GS 46

DANNY DOTNET
But I learned all that, Uncle Bob. It took me dozens and dozens of games, but I learned.

47 OFFICE 47

Note: P Chessboard

UNCLE BOB
Of course you did, Danny. It took time, but by playing a lot, and losing a lot, you learned enough strategy and tactics to play a pretty fair game.

48 GS 48

DANNY DOTNET
Yeah, I got pretty good. But I got a lot better after I read that book.

49 OFFICE 49

Note: P chessboard

UNCLE BOB
And that's the way that works. Raw experience can give you enough knowledge and skill to do a fair job. But if you want to be a master, you have to study the games, strategy, and tactics of past masters. You have to read the books. You have to master the patterns.

50 FAMILY ROOM 50

UNCLE BOB
And that's what the Design Patterns book is. It's a summary of the knowledge of past masters. It's a condensation of years of experience. Each pattern is a solution that past masters have used to resolve issues in their software.

51 GS 51

DANNY DOTNET
Wow, I never looked at it quite like that.

52 CAR 52

UNCLE BOB
And that’s why I say that the Design Patterns book is probably the most important book written about software in the last thirty or so years.

Segment 4 - The GOF book

53 WORKSTATION 53

Note: J GOF.jpg

Note: A highlight in order 2,3,1,4

UNCLE BOB
We call this the GOF book. GOF. G. O. F. It stands for “Gang of Four” and refers to the four authors: Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

54 KITCHEN 54

Note: 2

UNCLE BOB
The GOF monaker was awarded to them by those of us who were reviewing the chapters in 1993. This book was one of the first books to appear on the internet before it was published. The authors uploaded postscript files to a special FTP site on a daily basis. Those of us who reviewed them eagerly awaited each upload. We would read the uploaded chapters and then discuss the topics in an email group.

UNCLE BOB (CONT’D)
Soon we began referring to the authors as the gang of four; which was an oblique reference to the four Chinese Communist Party officials who came to prominence during the Cultural Revolution in the ‘60s and ‘70s and were subsequently charged with a series of treasonous crimes.

55 GS 55

SPOCK
Illogical.

56 BASEMENT BATHROOM 56

Note: A [GOF94].

UNCLE BOB
Yes, it was a politically incorrect joke; but it stuck. Nowadays bibliography citings of the book refer to it as [GOF94].

57 WORKSTATION 57

UNCLE BOB
The book begins with a chapter that both justifies and explains the Design Pattern concepts, and also presents the design principles that drive those patterns. In this chapter you will learn why it is important to design to an interface instead of an implementation, and why you should prefer composition to inheritance.

You'll find the discussions in this chapter to be both enlightening and enjoyable.

58 SUN ROOM 58

UNCLE BOB
Chapter 2 presents a case study of the design of a word processor. It shows how patterns were identified and used to solve the interesting problems presented by that problem. Again, you'll find this chapter to be illuminating and fun to read.

59 RECROOM 59

UNCLE BOB
(walking)
Then begins the catalog of patterns. This catalog is divided into three chapters which focus on the three broad categories of patterns: Creational, Structural, and Behavioral.
(MORE)

UNCLE BOB (CONT'D)
In this series, we will not be walking through these patterns in the order presented in the GOF book. Rather, we'll try to couple the patterns we talk about here, with the patterns being used in the case study series.

60 GS 60

MONK
Creational patterns are patterns that help you create instances of objects.

61 GS 61

DATA
Structural patterns help you to set up the communication pathways between separate groups of objects.

62 GS 62

SPOCK
Behavioral patterns help you partition the behaviors of your system into discrete sets of classes.

63 WORKSTATION 63

UNCLE BOB
But despite these classifications, all of these patterns pursue the same goal -- dependency management. Each one of these patterns is a crystalized application of the SOLID principles. Each one of them helps you to separate and decouple the various concerns within your system. And _that_ is why they are called _Design_ patterns.

Segment 5 - The Command Pattern

64 GS 64

RUBY ROD
Like, man, yknow, this is really fascinating and all; but like can you actually show us a pattern?

65 WORKSTATION 65

UNCLE BOB
Sure Ruby. For the rest of this episode, we're going to focus on one my favorite design patterns. The Command Pattern.

66 WHITEBOARD 66

UNCLE BOB
Of all the design pattern out there, Command is one of the simplest. Most often it is nothing more than an interface named Command with a single method named execute.

67 GS 67

JERRY JAVA
Yeah, yeah, execute, that's really procedural.

68 GS 68

RUBY ROD
Yeah, like, I mean, that's just a function man. I thought Design Patterns were all about objects and stuff.

69 KITCHEN 69

UNCLE BOB
Don't scoff at functions, guys. All objects contain functions.
(MORE)

UNCLE BOB (CONT'D)
But sometimes those functions need to be decoupled in unorthodox ways -- and that's where the command pattern comes in.

70 GS 70

DANNY DOTNET
Gosh, Uncle Bob, what do you mean by unorthodox decoupling?

71 MASSAGE CHAIR 71

UNCLE BOB
So let me tell you a story. Back in Episode 12, the episode about the interface segregation principle, I told you about my time as a consultant for Xerox, working on the control software for the first digital color copier.

72 GS 72

UNCLE BOB
To control this copier we had to start and stop motors, engage and disengage clutches, close and open relays, and, in general, control a large number of actuators.

73 WHITEBOARD 73

UNCLE BOB
One of the design options we played with was to use the Command pattern for each actuation. So we would derive a MotorOn command, a MotorOff command, a ClutchEngage command, a RelayClose command, and so on. We could create instances of these commands and pass them to other objects as instances of Command. For example, we had another class named Sensor that held a reference to a Command.
(MORE)

UNCLE BOB (CONT'D)
When the sensor was triggered (usually by a piece of paper moving over an optical sensor) the sensor object would call execute on it's command.

74 GS 74

UNCLE BOB
The interesting thing about this is that the sensors had no idea what command they were executing. The fact that the sensor controlled a relay was something that the sensor did not know.

The Command pattern allowed us to decouple WHAT was done, from WHO did it.

75 GS 75

SPOCK
Decoupling actions from actors seems likely to greatly enhance flexibility.

76 MASTER BATH 76

UNCLE BOB
Absolutely, but decoupling what from who is just the beginning of the Command Pattern's ability to decouple.

77 WHITEBOARD 77

Note: J employee class diagram.graffle

Note: J AddEmployeeClassDiagram.graffle

UNCLE BOB
Take, for instance, the payroll application from Episodes 7 and 18. Remember all those interactors that encapulated the use cases? Well the interactors were just instances of the Command pattern, weren't they?

78 GS 78

Note: J FactoriesBuilders.graffle

ENGINEER
Dang betcha they were. All we did was fetch ‘em from a factory and then execute ‘em. So, yep. All them use-case interactors were commands, jus’ like ya said.

79 WHITEBOARD 79

UNCLE BOB
So, now, Imagine that we have a web based GUI that allows users to add, change, or delete employees from the database. That GUI would create instances of those commands, wouldn’t they.

80 GS 80

Note: J Architecture.jpg

DANNY DOTNET
That’s right. Users would enter data into web forms, and then the controllers would use the appropriate factory to create the correponding interactor - er - command. And then the controller would execute that command.

81 EXERCIZE BIKE 81

UNCLE BOB
Right. But now imagine that the controller did everything but that last step. Imagine that instead of executing the command, the controller put the command instance into a list and left it unexecuted.

82 MASTER BEDROOM 82

Note P in bed. Lights out.

UNCLE BOB
And then. At midnight, we have some scheduled process walk through that list executing all those commands.

83 GS 83

RUBY ROD
Woah, like you decoupled what was done from WHEN it was done. That's like a temporal decoupling!

84 GS 84

Note: A K9

DOCTOR WHO
A Temporal Decoupling? K-9 remind me to check the temporal couplers the next time we land.

K9
Affirmative Master.

Segment 6 - Do and Undo

85 GS 85

DANNY DOTNET
OK, Uncle Bob, I can see that by using the Command pattern, we can decouple the behavior of a task from who invokes that task; and also from when that task is invoked. That's pretty cool. But is there more?

86 WORKSTATION 86

UNCLE BOB
There sure is! Consider the fact that instances of the command pattern are objects; and objects have state. When you call execute on command objects, they could record details about what they did. Now what do you think you could you do with that information, eh?

87 GS 87

SHERLOCK
What indeed! Obviously an object that remembers the relevant details about the actions it preforms; has all the information it needs to undo those actions at a later time.

88 MASSAGE CHAIR 88

UNCLE BOB
Precisely. Command objects are the perfect way to implement undo functionality. And that reminds me of another story.

89 GS 89

UNCLE BOB
Long ago I took a contract to develop a software system that involved the creation of 18 different desktop GUI applications.
(MORE)

UNCLE BOB (CONT'D)
They were all specialized drawing applications meant to help architects draw building plans, property lines, roof plans, landscaping plans, structural integrity plans, among many other things.

90 WHITEBOARD 90

Note: D Wire-frame diagram of GUI.

UNCLE BOB
The applications were pretty traditional 90s style palette and canvas applications. The palette consisted of a set of buttons down the left side of the screen. The canvas was at the right. Users would click on the palette buttons to choose a function like “Draw Window”, and would then click on the canvas to place that window into their drawing.

91 MAC 128 91

UNCLE BOB
Among the buttons in the palette were buttons like move and erase. If you wanted to move an object around on the screen, you’d click on the move button and then click the element on the canvas you wanted to move. If you wanted to erase an object, you clicked on the erase button and then on the object you wanted to erase.

92 GS 92

JERRY JAVA
Yeah, yeah, that sounds kinda clunky.

93 KITCHEN 93

Note: P eating sandwich.

UNCLE BOB
Hey! What can I tell you? It was the ‘90s. Jeez!

94 OFFICE 94

UNCLE BOB
Anyway, one of the buttons on the palette was the undo button. When you clicked that, it would undo the last operation the user performed. So if the last operation was to place a door, or a window, or draw a property line, clicking undo would erase what you had created. If your last operation was to erase or move an item, clicking undo would restore that item to it’s original position.

95 CAR 95

UNCLE BOB
And, of course, you could click undo as many times as you liked. It would simply keep undoing previous operations all the way back to the beginning of time.

96 GS 96

DOCTOR WHO
I been there. It’s a pleasant enough place to visit. Though I can’t recommend the night life.

97 GS 97

DANNY DOTNET
Gosh, Uncle Bob, did you use the command pattern to implement undo? How did that work?

98 WORKSTATION 98

UNCLE BOB
We did, indeed, use the command pattern Danny. It worked like this.

99 WHITEBOARD 99

UNCLE BOB
We wrote this in C++, which doesn't have interfaces. So we created an abstract class named Task and gave it an abstract method named execute.

100 GS 100

RUBY ROD
Hey, man, like why didn't you name it Command? I mean, I thought the names were so important and all.

101 GS 101

UNCLE BOB
The names _are_ important, Ruby. But we wrote this code long before the Design Patterns book was published. So we didn't have a standard name to use. If we were writing that application today, we'd certainly name that class Command.

102 WHITEBOARD 102

UNCLE BOB
Anyway, we created derivatives of the Task class for each of the functions on the palette. Every palette button corresponded to one of the task derivatives. When you clicked a button on the palette, our system would create the appropriate task instance, and then execute it.

103 GS 103

DANNY DOTNET
OK, I think I understand. If the user clicked the 'Add window' button, you'd create the AddWindowTask and then execute it right?

104 WORKSTATION 104

UNCLE BOB
Right.

105 GS 105

DANNY DOTNET
And if the user clicked on the 'Erase' button, you'd create an instance of the EraseTask and execute it right?

106 WORKSTATION 106

UNCLE BOB
Right.

107 GS 107

DANNY DOTNET
So then, how did you do Undo?

108 WHITEBOARD 108

UNCLE BOB
We added another method to the Task class. This method was named undo. When execute was called on a task that task would perform the appropriate action, and it would also store in it's fields, the information necessary for the undo function to undo that action.

109 GS 109

DANNY DOTNET
OK, wait, let me see if I've got this right. When the user clicked on the AddWindow button, the system created a instance of AddWindowTask, and executed it. The AddWindowTask allowed the user to click in the canvas where they wanted the window. The task would add that window, but would also remember that it had added that window.
(MORE)

DANNY DOTNET (CONT'D)
Later, if you called undo on that task, it would find that the window it remembered, and erase it. Right?

110 WORKSTATION 110

UNCLE BOB
Yes, right.

111 GS 111

DANNY DOTNET
Or if the user clicked on the Erase button, you'd create an instance of the EraseTask and execute it. The Erase Task allowed the user to click in the canvas on the item they wanted to delete. The task deleted that item, but also remembered that item, and it's location on the canvas in some of it's fields. If you called undo on that task, it would put the remembered item back where it was, right?

112 WORKSTATION 112

UNCLE BOB
Yes, that's exactly right. I think you understand it quite well.

113 GS 113

DANNY DOTNET
Hee hee.

114 GS 114

RUBY ROD
Yeah, that's cool and all; but, like, what about redo?

115 WHITEBOARD 115

UNCLE BOB
Well, an object that has undone an action, can store the information necessary to redo that action, can't it.

116 GS 116

RUBY ROD
Yeah, but then like what if the user does five things, and undoes three, and then redoes one and then does something else, and then redoes, and, like that could get really confusing.

117 GS 117

DATA
I believe it would be wise, when implementing redo, to keep close control over the stack of executed command objects; lest they get badly out of hand.

Segment 7 - The Actor Model

118 GS 118

RUBY ROD
OK, so like, with the Command Pattern we can decouple what from who, and what from when; and we can implement undo and redo; and that's a lot. Is there, like, anything else.

119 WORKSTATION 119

UNCLE BOB
Well, there's always the actor model.

120 GS 120

DARTH VADER
To be, or not to be. That is the question. Whether tis nobler....

TV STATIC

121 MASSAGE CHAIR 121

UNCLE BOB
So, I think we have time for one more story, don't we?

122 WORKSTATION 122

UNCLE BOB
OK, so, back to that contract I had with Xerox in the early '90s. We were using C++ to control the internal hardware of a color copier. And, as I told you before, there were lots of devices to control. Motors, clutches, relays, solenoids, you name it. These devices had really strict real-time deadlines; and they all had to be controlled asychronously. And that meant threads. Lots and lots of threads.

123 MAC 128 123

UNCLE BOB
Now the problem with threads is that, in most cases, each thread has to have it’s own stack. And that means you have to allocate memory for every stack for every thread. Now if you’ve ever done this, you know that the most important question is, how much memory to allocate for each stack.

124 WHITEBOARD 124

Note: D memory map of lots of stacks.

UNCLE BOB
The reason that’s important is that, on the 68000 microprocessor that we were using, there was no good way to detect if a thread had exceeded the capacity of it’s stack. Now imagine all these stacks in memory. If a thread did exceed the capacity of it’s stack, it could overwrite the data on some other thread’s stack. That poor thread, whenever it’s turn came to resume execution, would likely behave erratically and unpredictably.

125 GS 125

JERRY JAVA
Yeah, yeah, like Boom!

126 GS 126

SPOCK
In light of the fact that a stack overwrite like this has recently been determined as the cause of the uncontrollable rapid acceleration of certain automobiles, the word “crash” might be more appropriate.

127 OFFICE 127

UNCLE BOB
Or maybe not. This is serious stuff. Stack overwrite defects have killed innocent and unsuspecting people who were simply trying to drive their cars.

128 GS 128

DANNY DOTNET
Gosh, Uncle Bob, so how much space do you allocate for a thread's stack?

129 MAC 128 129

UNCLE BOB
That's a tough question to answer. At first you might think that you could simply count all the function call arguments, and all the local variables, and all the function call stack frames in the thread, and determine the most possible stack usage. But then you have to remember that there are interrupts running, and those interrupt heads use the stack of the currently running thread. Those interrupts might just happen at the moment where your thread was at maximum stack usage. What's more, it's possible that one interrupt could be interrupted by another, higher priority, interrupt.

130 GS 130

RUBY ROD
Woah, that sounds like really complicated. So why don't you just make the stack really really big.

131 COUCH 131

UNCLE BOB
Yeah, well, in fact, that's what most people do.
(MORE)

UNCLE BOB (CONT'D)
They simply allocate unreasonably large stacks to make sure they have a big safety buffer -- and then they cross their fingers.

132 GS 132

SHERLOCK
A decidedly questionable practice is you ask me.

133 WHITEBOARD 133

UNCLE BOB
Quite. But even that option wasn't open to us, because we needed hundreds of thread. And in an embedded machine with limited RAM, you simply cannot allocate huge stacks for hundreds of threads. You don't have the memory.

134 GS 134

DANNY DOTNET
Gosh, Uncle Bob, so what did you do?

135 GS 135

UNCLE BOB
We used the actor model, Danny. The actor model is a simple idea that has been used for decades in systems that require hundreds or thousands of threads in a constrained memory environment. It works like this.

136 WHITEBOARD 136

Note: D actor model code on board.

UNCLE BOB
Look at this code. You see the Command interface up at the top? You see the list of commands?
(MORE)

UNCLE BOB (CONT'D)
You see the run method with that loop in it. Look carefully now. What does that run method do.

ZOOM IN ON BOARD
AND FREEZE

137 GS 137

Note: A picture in picture on top of last frame of previous scene.

DANNY DOTNET
Gosh, Uncle Bob, it just executes every command in the list and then exits.

138 WORKSTATION 138

Note D: actor model code on screen

UNCLE BOB
OK, so if you put one command in the list and called run, how many commands would be executed before run returned?

139 GS 139

DANNY DOTNET
One.

140 WORKSTATION 140

Note D: actor model code on screen

UNCLE BOB
Are you sure about that? Is there a way that more commands could be executed than just that one?

141 GS 141

DANNY DOTNET
Well, gosh, Uncle Bob, there's only one command in the list. If you call run, only one command will be executed. Right?

142 WHITEBOARD 142

Note: D actor model on board

UNCLE BOB
Well, Danny, what if the execute method of the command in the list, puts another command in the list?

143 GS 143

DANNY DOTNET
Oh. Oh! OH! Gosh, Uncle Bob, if you did that, then _anything_ could happen. I mean, the run method might _never_ return. It could create dozens and dozens of new commands, and each of them could create even more, and... Oh wow!

144 WORKSTATION 144

Note: D ButtonCommand and Light Command code on screen.

UNCLE BOB
Wow indeed. So, now, imagine that I have a command named ButtonCommand. When execute it called it looks at the state of a button. If that button has not been pressed, then the execute method puts itself back on the list.

But, if the button _has_ been pressed, then the execute method create an instance of LightCommand and puts that on the list.

The execute method of the Light command simply turns on a light and then returns.

Now, if you put just that one ButtonCommand on the list, and then call run, what will it do?

145 GS 145

RUBY ROD
Man, like, it'll wait for you to push the button. When you do, the light will turn on, and then the run method will return.

146 MAC 128 146

UNCLE BOB
Right. In the actor model, threads wait by having a command check for an event that puts itself back on the list if that event has not occurred.

147 GS 147

JERRY JAVA
Yeah, yeah, that doesn't look much like a thread to me.

148 GS 148

Note: D new ButtonCommand code

UNCLE BOB
Well then consider this. I change the ButtonCommand constructor to take two arguments. The first is the io address of the button to check, the second is a command to execute if the button has been pressed.

I create five button commands, one for each of five different buttons, each with a light command that will turn on a different light.

I call run. What happens?

149 GS 149

DANNY DOTNET
The system will wait for all five buttons. Every time you push one of those buttons, the corresponding light will go on.

150 WORKSTATION 150

UNCLE BOB
Let me stop you right there. Yes, you're right. As soon as you push a button, the corresponding light will go on. But then, how many commands are left in the list.

151 GS 151

RUBY ROD
Four. There's only the four ButtonCommands left. The LightCommand just exited.

152 WORKSTATION 152

UNCLE BOB
Right. So the thread for that button and light, died.

153 GS 153

DANNY DOTNET
Oh! I get it, I get it. There were five threads at first, and one of them just died. So every time you push a button, the corresponding light will go on, and the thread for that button will die. When the last thread dies, the list will be empty, and the run method will return.

154 GS 154

JERRY JAVA
Yeah, yeah, but I don't see any threads.

155 GS 155

DANNY DOTNET
Well there aren't any real threads, like a Java thread. But there is a sequence of command executions that are threaded together. That sequence is kind of like a thread.

156 WORKSTATION 156

UNCLE BOB
Right you are! The thread is nothing more than the sequence of command executions. And you can make that sequence as simple, or as complex as you like. And since the commands for all the threads are in the list, the execution of those threads is asychronous.

157 GS 157

RUBY ROD
Heh, that's really cool.

158 GREAT ROOM 158

UNCLE BOB
It gets better. What stack are all those threads using?

159 GS 159

DANNY DOTNET
Oh, gosh, wow, Uncle Bob, they're all using the same stack that the run method uses -- just the one normal system stack.

160 GREAT ROOM 160

UNCLE BOB
Right! And why is it that those threads can all use the same stack?

161 GS 161

RUBY ROD
Well, because the commands are just functions, I mean, they all return back to the run method without blocking or waiting.

162 GREAT ROOM 162

UNCLE BOB
Right. And because of that, this type of threading is sometimes called the Run to Completion model, because all the commands run to completion before any other command can run.

163 GS 163

JERRY JAVA
Yeah, run to completion, so all those threads run, and yet there's just one stack?

164 GREAT ROOM 164

UNCLE BOB
Right. And that means we can have hundreds, even thousands, of threads, without allocating separate stacks for them. We can keep our stack very large, without fear of crashing some other thread.

165 GS 165

SPOCK
Simple and Logical. It would seem advisable for the manufacturers of automobiles to consider using the actor model for their embedded software systems.

166 OFFICE 166

UNCLE BOB
Indeed it would. After all, the telecommunications industry has been using this model for their switching systems for decades.

Segment 8 - Conclusion

167 WORKSTATION 167

UNCLE BOB
So that's it. That's the episode on the Command Patterns, and the introduction to Design patterns.

168 GS 168

JERRY JAVA
Yeah, yeah. We started out by defining what design patterns were. Solutions to problems in contexts.

169 GS 169

DANNY DOTNET
Then we walked through the structure of the GOF book, and encouraged everyone to read it carefully.

170 GS 170

RUBY ROD
Yeah, man, and then we, like talked about the structure of the Command Pattern, and how it was good at decoupling what from who, and what from when; and, man, that's a lot of w's.

171 GS 171

DATA
Then we discussed the utillity of the command pattern for implementing undo and redo.

172 GS 172

SPOCK
And finally we looked at how the command pattern can be used to implement the actor model, allowing systems with large numbers of asychronous threads to share a common stack.

173 OFFICE 173

UNCLE BOB
Before we close, let's take a moment to reflect on the fact that people have died because of improperly constructed software. The more our civilization depends on the software we write; the more people's lives will hang in the balance.

174 FRONT DOOR 174

Note: p With dogs.

UNCLE BOB
And with that, it's time to close the episode. But we've still got loads and loads to talk about. There are a couple of dozen design patterns yet to discuss; and then there's professionalism, acceptance testing, functional programming, and more. And keep an eye out for that case study series. You won't want to miss the next exciting episode of Clean Code. Factory Patterns. Come on you dogs.