

Storyboard For *Director* Presentation On My Approach Towards Training A Concept In *Perl*

Barry Krusch

Note to reader: several clients have asked to see storyboards I have prepared, so I thought I would prepare a non-proprietary storyboard I could display on this web site for a brief *Director* rich-media training course on how I would teach a key concept in the language of *Perl*. My prospective audience for this presentation would be . . . you! The storyboard uses various media assets, including sound (WMA), video (MPEG), *Photoshop* (PSD), and *Flash* (SWF) files.

The following document illustrates four things:

1. How I prepare storyboards;
2. My overall instructional design strategy;
3. My approach towards teaching technical material;
4. How I illustrate concepts visually (i.e., how I employ rich media content to communicate).

Right now this project only exists as a storyboard, but if I ever get some spare time I will convert it into an actual presentation . . . one day!

VISUAL					SCRIPT
SEQ #	TRANSITION	CONTENT	ASSET TYPE	ACTION	
1		Title: Teaching <i>Perl</i> : My Approach	Video (MPEG) JumpBack with voiceover 3-D letters? Title Overlaid in After Effects	Perhaps have 3-D instantiation of the line of <i>Perl</i>	Since I've been working in the field of e-learning for over a decade now, I thought you'd want to see an example of how I'd approach the task of training a novice learner in a highly complex technical area. The main idea I want to convey in this short segment is that no matter how complicated the training job, someone can be brought from a state of ignorance to expertise (in the shortest possible time) by using various tricks of the trade.
2	Wipe	Educational Heuristics	PSD	Drop from top	The official name for these tricks is <i>educational heuristics</i> . Now, there are dozens of these learning tricks, but I'm going to focus on only 3 here, which are
		Situate... context	PSD	Fly from right	1, situate all learning within a real-world context,
		Simplify... task	PSD	Fly from right	2, simplify the learning task, and
		Introduce... degrees	PSD	Fly from right	3, introduce changes by degrees.
3	Wipe	s <[!> <.>;	PSD	Wipe out heuristics. Special effect.	I'm going to illustrate these techniques with this example of source code from <i>Perl</i> , which is, after <i>Java</i> , the Internet's favorite computer language.
					Now, don't get nervous; you'll soon see this code's bark is worse than its bite. Suppose my task is to take a student who knows nothing about <i>Perl</i> and give them an understanding of this gobbledygook in less than 5 minutes. How would I tackle this unappetizing chore?
					Well, let me start by telling you what I would <i>not</i> do, which is teach this material using the standard instructional design technique of orientation by classification.
4		s <[!> <.>; (with labels)		Fly in labels	For example, I could begin by pointing out that the entire line is called a <i>regular expression</i> , and next proceed by labeling the sub-parts of the line; in other words,
					this is a <i>delimiter</i> ,
					this is a <i>character class</i> ,
					this is a <i>statement termination indicator</i> , and so forth. Sure, I could label all these parts with the appropriate jargon, but after I was done you would have no more knowledge of what was really going on in this line than when you began.

5	Direct Transition	In <i>Perl</i> , the < symbol is an example of a 1) quantifier 2) delimiter 3) statement termination indicator 4) character class			If I was extra cruel, I'd then follow up this learning session with a multiple-choice exam to see if you knew what label went with what part. Good luck! And are you ready for this irony? Even if you got the right answer, you <i>still</i> wouldn't know the purpose of the line. So what have we accomplished? Clearly, an alternative approach is needed.
6	Wipe to blank.	Situate... context		Slide in first heuristic: this gets situated at top left. Fade out code with labels.	Here's the approach I'd take: I would utilize the first educational heuristic I discussed a little while ago, and begin not by labeling the code's various parts, but by supplying real-world context to provide background and motivation: in other words, we're not just going to be learning for the sake of learning here, memorizing a whole list of vocabulary words and then getting quizzed on them: instead, we'll be learning to solve a problem, a business problem, a problem which might pop up in one's day-to-day work life. So instead of starting from the code, I'd start from a <u>scenario</u> , which might go something like this:
7	Wipe	[See script]	SWF	Slide out code and slide in scenario. Special effect: this appears under heuristic. Bring in photo of workplace.	<i>You have a client whose last name was spelled 'Jonson', without the 'h', but you inadvertently misspelled the name of the client in an e-mail as 'Johnson', with the 'h', and a number of members of your team in this very large project used that spelling, and now there are 75 memos which need correcting before one of them inadvertently gets sent to the client. How can you fix the spelling of the client's name in these memos without having to open each one separately?</i>
		Simplify... task		Slide in second heuristic to upper middle.	Now, you'll note that by creating this scenario, I simultaneously simplified the learning task, and this act of simplification is, as you'll recall, our second heuristic. You see, the original example was too complicated to easily explain.
8		s <[!> <.>;	PSD	Slide out scenario and photo, slide in simpler example.	Remember this? Teaching, when centered around learners, proceeds in stages, rung-by-rung, and the first rung is the simplest possible. So instead of starting by teaching the final thing we want our learners to comprehend, we'll start by teaching a much simpler example related to our scenario, which is how you can change one spelling of Johnson to another using <i>Perl</i> .
		s <[!> <.>;	PSD	Fly out individual characters. Slide delimiters to leave room for new material.	From this,
		s <Johnson> <Jonson>;	PSD	Slide in names.	to this.

9		<code>s <Johnson> <Jonson>;</code>			Note that when you have a clear objective, the key aspects of this line of <i>Perl</i> are magically illuminated. In our scenario, we wanted to change all spellings of “Johnson” with an ‘h’ to “Jonson” without the ‘h’, and it’s easy to see that happening in this line.
		<code>s <Johnson> <Jonson>;</code>	Substitute this... with this.	Label parts	Once you learn that the letter s in <i>Perl</i> as used here means <i>substitute</i> , you can easily understand that this line of <i>Perl</i> is basically saying “substitute every example of the words in the <i>first</i> set of angle brackets with the words in the <i>second</i> set of angle brackets”.
			Indicates end of line.		And the semi-colon at the end simply tells <i>Perl</i> that you’re ending a line. Seems a lot clearer now, doesn’t it? With this major insight under our belts, we can now move quickly to an understanding of the original line by moving to other cases: for example, if you wanted to change <i>Jones</i> in a document to <i>Smith</i> , what would you write?
10	Wipe	<code>s <jones> <smith>;</code>			You would write this: And if you wanted to change <i>young</i> to <i>old</i> ,
	Wipe	<code>s <young> <old>;</code>			you would write this:
11		<code>s <?> <!>;</code>		Slide in third heuristic, upper right. Slide out young/old, slide in punctuation.	Having looked briefly at these additional examples which anchored the point, let’s now move a little closer to our original example, using our third and final heuristic of introducing changes by degrees. Suppose you wanted to change a question mark to an exclamation point? You’d write this:
12		<code>s <?> <.>;</code>		Fly up !, fly . in place	And suppose you want to change a question mark in the document to a period? You’d write this:
					If you’ve got a good memory, you can see that we’re just one step away from that which initially seemed so forbidding, our original example. Let’s go to that now. Suppose instead of wanting to change only question marks to a period, you wanted to change question marks <i>and</i> exclamation points to periods. If that was the case, then you would put those two pieces of punctuation in between brackets,
		<code>s <[?!] <.>;</code>		Slide delimiters, move in stuff	like this. The official name for this is a <i>character class</i> , which you use every time you have more than one character you want to change. And here we have our original line, which simply says “if you see a question mark or exclamation point, change it to a period”. And that’s it!

13		Summary	Video MPEG jump back	<p>Of course, as those familiar with <i>Perl</i> know, I've left out a great deal of information regarding how the line would be used in the scenario, not to mention other information about how the line would appear in a <i>Perl</i> script designed to solve the problem posed by the scenario. And of course I could enhance and develop what I taught here even further. However, my objective in this short segment was not to give a complete explanation of the language of <i>Perl</i>, but merely to show my approach towards training this short piece of material, which would then reveal the methodology I would employ on larger projects. I hope you now have a good understanding of this methodology.</p>
----	--	---------	-------------------------	---