from
June 2014
Drupal 8

DRUPL'ART

Patterns

Doug Green

0

I like *new* movies and *old* music.

Why new movies but old music?

Maybe new TV is better because old movies – and TV – were not very good simply because moving pictures were a new medium, and it has taken artists time to mature.

Maybe old music is better because rock 'n' roll of the ‘60’s and ‘70’s was particularly brilliant, a musical Renaissance era.

But we do know that art is subjective; what I like is not necessarily what you like.

So though I prefer contemporary movies over the classics, some people opt for the opposite; they think old TV shows were the Renaissance period – and I just have no taste for good television.

Is there possibly another explanation, having to do with *repetition*?

Does repetition strengthen and reinforce one’s preference? Or do we just repeat things we like... a lot?

I often watch every episode and every season of a TV show I enjoy, but I watch each episode only once. Each episode is similar, but different. The patterns of the characters and interactions are similar, but the dialogue and exact story is, of course, different.

But with music, not only do I listen to the same artists, I listen to the same songs over and

over again. I've heard "Jailhouse Rock" a million times, but only watched the movie once or twice. Hearing something I like, repeatedly, reinforces the song's pattern, makes me more comfortable with it, and, I think, ultimately leads me to liking it more.

The same could be said of software patterns.

Building on my last article on "Sculpting Conditionals," nothing helps me more than reviewing someone else's code. I can quickly detect a pattern I'm not familiar with. Frequently, an unfamiliar design pattern indicates trouble. And if the bad pattern occurs once in a code review, it's worth checking to see if it was done elsewhere in code that is already committed. Unfamiliar patterns should be a klaxon horn, something deserving attention. After some study I might discover that the new pattern is actually better than the old pattern, and it will become something I adopt, but it is always initially a warning.

Great developers can create new patterns. Good developers can learn from those patterns and become great. Novice developers can become good developers simply by following the patterns found in best practices.

I've waged an internal debate over the advantage of Object Oriented languages OOP verses simple OO concepts for years. Good developers can (mostly) create OO programs with a traditional language. Novice developers can also create OO programs if they follow an OO pattern, though following a pattern is only a convention.

I've concluded recently that the advantage of OOP – and the advantage of Drupal 8 – is that OO is now not just a pattern to be followed, but is enforced by the language. OO as a pattern can be ignored, but OOP as a language, must be followed. I hope that Drupal 8's move to OOP will lead to fewer production bugs because the language will force us to find mistakes in development.

Before we all get the goodies in Drupal 8, here are a few Drupal 7 patterns you should learn to repeat: text translation, string sanitation, string concatenation, the global user object. I use them because they are patterns that I recognize and require little extraneous thought. You should use them because they are community-tested – and they just work.

Anywhere I see an English string, I think about translation, wrapping the text with `t()`. There are some exceptions, notably in `hook_menu()` but other hooks deserve a quick check on api.drupal.org. Text inside a link is a common pattern that actually reinforces the argument order, `l(t(), ...)`

Anywhere I see a title, I think about `check_plain()`. And if I see a title being rendered, I wonder if other data alongside it might also need sanitation.

Anywhere I see output using `theme()`, I wonder if the result can be returned in a renderable array. Renderable arrays are wonderful Drupal 7 innovations that delay the theming of output until the very last moment, and, in the case of caching, delays the rendering of output entirely. If you call `theme()` directly, you are likely doing something wrong. Page (menu), form, and block callbacks should return arrays, not strings. Drupal 8 has taken this to the next level, by removing the `theme()` function entirely.

I also love render caching. In the render array, use `#pre_render` to build your data, `#cache` to save the data, and `#theme` to display it.

The global `$user` object is a source of security errors. Many a developer has accidentally overwritten or changed data on the global object, giving the current user more privileges than they deserve. Two simple patterns usually solve this problem: First, use `$GLOBALS['user']` instead of `$user`. This simple change comes with a mindset that is read-only. Second, always name local variables that hold user data `$account`, never `$user`.

There are many other good patterns. The best recommendation I can offer is to read and copy code that is in core. While Drupal core is not perfect, it has been community-tested and offers the best that many Drupal minds can offer.

Image: ©iStockphoto.com/deveritt (http://www.istockphoto.com/deveritt)