Dependency injection is not a virtue

By David Heinemeier Hansson on Jan 6, 2013

In languages less open than Ruby, hard-coded class references can make testing tough. If your Java code has `Date date = new Date();` buried in its guts, how do you set it to a known value you can then compare against in your tests? Well, you don't. So what you do instead is pass in the date as part of the parameters to your method. You inject the dependency on Date. Yay, testable code!

As has unfortunately happened with a variety of patterns that originate from rigid languages like Java, Dependency Injection has spread and been advocated as a cross-language best practice on trumped up benefits of flexibility and malleability. If your code never knows exactly who it's talking to, it can talk to anyone! Testing stubs, mocks, and future collaborators. Hogwash.

Ruby is like Play-Doh not LEGO and the ways it'll bend to extract the best of hard coding is truly impressive. Take the publish! example from Adam Keys' Design for test vs design for API:

```
def publish!
  self.update published_at: Time.now
end
```

In less open languages that's an obvious problem to test. But in Ruby it couldn't be easier:

```
Time.stub(:now) { Time.new(2012, 12, 24) }
article.publish!
assert_equal 24, article.published_at.day
```

This pattern is so useful that Travis Jeffrey wrapped it up in the popular Timecop gem. But if you've acquired a design taste for the Java-friendly pattern of dependency injection, it looks gross. It's a gut reaction trained on pattern matching. The brain goes HARD CODE ALERT! HARD CODE ALERT! That's the danger with patterns: they can quickly graduate from

tool to taste.

Adam Keys' goes on to put forward a fine argument of whether you're designing for your tests or your API (and Lloyd Kupchanko also shared some wise words on how the API suffers from DI). I don't think there's actually much of a dichotomy between the two in a language like Ruby. We can have the clarity and simplicity of hard coded references and *still* be able to easily test them, as shown above.

Of course, this runs counter to another Java-derived principle to only mock types you own. Combine that with an affinity of dependency injection and the simplest thing that could possibly work is no longer not just good enough, it's disgusting. It *violates* the doctrine that has been so carefully assembled.

That's the real point here: Be careful with who you share your intellectual foundation with. It's fashionable to say "I'm not a Ruby programmer, I'm just a programmer". But languages shape the way we think. While we can cross-pollinate some ideas between languages, there are many we cannot. And worse, the incompatibility is not immediately apparent — especially when they both seem to just be Objective Oriented.

I'm a Ruby programmer.