Polishing Ruby

Musings on Ruby and the Ruby Community...

fat code is slow code

By zenspider on November 30, 2008 4:09 AM | Permalink | Comments (0)

I've been poking around in a lot of gems lately and I found this beauty (I added the 1 to the name for obvious reasons below):

```
def load_tags1()
  x = Array.new() # Group and Element ID
  y = Array.new() # Value representation
  z = Array.new() # Name
  # E.1 Registry of DICOM command elements
  # Group 0000
  x+=["0000,0000"] and y+=[["UL"]] and z+=["Command Group Length"]
  x+=["0000,0001"] and y+=[["UL"]] and z+=["Length to End"] # RET
  x+=["0000,0002"] and y+=[["UI"]] and z+=["Affected SOP Class UID"]
  x+=["0000,0003"] and y+=[["UI"]] and z+=["Requested SOP Class UID"]
  x+=["0000,0010"] and y+=[["CS"]] and z+=["Recognition Code"] # RET
  # ... 2500 more lines like the above
  # Return the array information:
  return [x,y,z]
end
```

This *one method* flogs at an astounding **15538**! As a comparison, my entire perforce repository flogs at 37391. The next highest method I've found flogged at 3685.

What's wrong with this code? Well, for starters, it flogs to 15k for a reason. That's a LOT of code and almost all of it need not exist to begin with. A very quick analysis shows that NONE of the `and` logic needs to exist. Those could be semicolons. Further, this is really all static data and the method returns the 3 columns... So why not treat them as columns and be done with the 3 variables? Here was my quick 1 minute rewrite (no, really):

```
def load_tags2()
  # E.1 Registry of DICOM command elements
  # Group 0000
  [["0000,0000", ["UL"], "Command Group Length"],
   ["0000,0001", ["UL"], "Length to End"],
   ["0000,0002", ["UI"], "Affected SOP Class UID"],
   ["0000,0003", ["UI"], "Requested SOP Class UID"],
   ["0000,0010", ["CS"], "Recognition Code"],
   # ... 2500 more lines like the above
  ].transpose
end
```

This returns exactly the same data. What does that flog to? 1. That's right, ***one***. Why? Because there is a total of 1 method calls throughout the whole thing. The rest is all static data.

And my final rewrite:

```
def load_tags3()
  TAGS
end
```

Just realizes that the whole thing is static/constant so you should treat it as such. This probably flogs at less than one. I didn't bother figuring out because once you go from 15k to 1 the rest is cake.

The real important part is this: **Flog scores mean something real**. In this case, flog complexity not only translates to "hard to test", it also means "hard to run":

```
% ./quick.rb 1000
# of iterations = 1000
                          user     system      total        real
null_time             0.000000   0.000000   0.000000 (  0.000152)
load_tags1           61.910000   0.340000  62.250000 ( 62.716426)
load_tags2            3.540000   0.020000   3.560000 (  3.575550)
load_tags3            0.000000   0.000000   0.000000 (  0.000504)
```

`load_tags1` is 17.5x slower that `load_tags2`, which is in turn 3580x slower than `load_tags3`.

It pays to flog.

Categories: Ruby, flog

Leave a comment

Sign in to comment on this entry.