Teuthida Technologies Home

All about embedded systems engineering.

Unconventional tools, #2

Posted on **October 7, 2016**

Hi All;

Now, we've all seen code that at least appears to work, but is very poorly written. You might even say the code "stinks". Now, it can be really hard to detect code smells in your own code. A major air freshener company speaks of people going "noseblind" to smells. The old solution to this problem was to have code reviews. This never works. People gloss over problems, the reviews take forever, and feelings can be hurt or axes can be ground. What is needed is an objective, easy-to-perform, review of the code. What is needed is an automated tool!

For me, programming in free ruby, the tool is called… reek!

Now you may think: "Why do I need a tool to complain about my working code?" I must admit, I sometimes think that too. The answer to that question is that whenever I dig into the code and clean up the smells, the result is almost always *much* better code. Now, I know that's a serious claim, so I am going to back it up with a real example from my own code. The following bit of code is used to process snippets of ruby code embedded in strings and surrounded by {{ }} structures. They generally go by the name "handlebars". Here is our starting point:

```
#Process a string with code embedded in handlebars and
#backslash quotes.
def eval_handlebars(in_str)
  out_str = ""

  loop do
    pre_match, match, in_str = in_str.partition(/{{.*?}}/m)
    out_str << pre_match
    return out_str.gsub(/\\\S/) {|found| found[1]} if match.empty?

    code = match[2...-2]
    silent = code.end_with?("#")
    result = instance_eval(code)
    out_str << result.to_s unless silent
  end
end
```

When I ran reek against this code, it made the following observations:

```
lib/mysh/user_input/handlebars.rb -- 2 warnings:
 [25, 27, 32]:FeatureEnvy: Object#eval_handlebars refers to
  out_str more than self (maybe move it to another class?)
 [19]:TooManyStatements: Object#eval_handlebars has approx 10
  statements
2 total warnings
```

Well, the code is pretty ugly, so perhaps this is not surprising. So; how to proceed? The first clue comes from the comment line.

```
#Process a string with code embedded in handlebars and
#backslash quotes.
```

This method is doing two distinct things! Maybe it should be two distinct methods? So here's the first fixup:

```

#Process a string with code embedded in handlebars and
#backslash quotes.
def eval_handlebars(str)
  do_process_handlebars(str).gsub(/\\\S/) {|found| found[1]}
end

private

#Process a string with code embedded in handlebars.
def do_process_handlebars(in_str)
  out_str = ""

  loop do
    pre_match, match, in_str = in_str.partition(/{{.*?}}/m)
    out_str << pre_match
    return out_str if match.empty?

    code = match[2...-2]
    silent = code.end_with?("#")
    result = instance_eval(code)
    out_str << result.to_s unless silent
  end
end
```

The new method do_process_handlebars just does handlebars while the original method takes that result and processes backslash quotes. So what does reek think of the code now?

```
lib/mysh/user_input/handlebars.rb -- 1 warning:
 [26]:TooManyStatements: Object#do_process_handlebars has approx 9
  statements
1 total warning
```

This is a lot better, but now my new method is too long! What to do? I could ignore the problem; I could tell reek to just ignore the problem; or I could mashup the code and use sneaky tricks to make it use fewer lines. None of these are good choices here. Instead, let's really look at what the code is doing. We see a loop, processing a string by repeatedly looking for a regular expression and then performing a substitution of the found text with new text. Wow! Did I really write that code? What we have here is a kludgy reinvention of the gsub method! The very same method already used to perform the backslash quoting. Zounds!

OK; so let's see what happens when we replace the kludge of gsub with that actual gsub method:

```
#Process a string with code embedded in handlebars.
def do_process_handlebars(str)
  str.gsub(/{{.*?}}/m) do |match|
    code = match[2...-2]
    silent = code.end_with?("#")
    result = instance_eval(code)

    (result unless silent).to_s
  end
end
```

Wow! That code is MUCH better looking! What does reek think about it?

```
0 total warnings
```

No more code smells found, well at least by the reek tool. I did make one further change. I corrected the ambiguous top comment line:

```
#Process a string with backslash quotes and code embedded
#in handlebars.
```

It would seem that even automated code scanning tools do not check for poorly written comments.

So, I think it is pretty clear that the new code is much better than the original. I can tell you that it also runs faster and uses less memory. Can we draw a conclusion from all of this? How about:

When things smell bad, put away the scented air spray and just clean up the mess!

Best regards;

Peter Camilleri (aka Squidly Jones)

Notes:

1. In the quest to write better code, I am inspired by Sandi Metz. An awesome video by her on the matter of smelly code is Get a Whiff of This by Sandi Metz.
2. Some code was slightly reformatted to fit into the blog post.

This entry was posted in **Dev Tools** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=1761]** .