Finding of the Week

Context and and intention are essential

Onkobu Tanaake

2020-04-15T10:00:00

I really enjoy taking part in other developer's struggles. This includes playing developer's games. One of them is the so called Finding of the Week. The intention is to educate, relief from stress and to remember the early days of programming. A group of developers comes together online. And everyone presents a piece of code and a very (geek) entertaining discussion starts. Once this turned awkward for me and implicitely many others.

The rules are very easy: bring a strange piece of code and a question or task. It depends on the players how this evolves and sometimes becomes a deep dive into the code dungeon. This is the most entertaining mode for me. We open our vaults, stores and visit other's attics to find the ugliest piece of code. While playing this mode a footer would be necessary reading »Don't try this at home.« This entertains all to the same degree but requires all to be experts/ seniors.

A more common mode is to ask why the short snippet doesn't work. It is essential for the one who asks to have absolutely no programming skills and to ask a group of want-to-be-experts. This is only entertaining for the others and never causes any insights but only misunderstandings. It is spiced up with a steep decline to more and more personal insulting comments.

Another mode is pure entertainment, e.g. with a comic strip. A very good source is xkcd.com. The best example of coder's edutainment is Bobby Tables: xkcd.com [https://xkcd.com/327/]. Also take the medium into consideration. Such a comic strip is only for the laughter and very abstract. There are still developers out there writing code that suffers from the depicted SQL injection vulnerability. It is much harder to defend yourself, when such a coding style costs real money. Depending on the amount they'll find the coder in charge.

So I prefer to ingest a lot of these show-and-tells or findings of the week. I also reflect upon my way of reviewing code and doing the real educating part. Again the medium is very important. Sitting at another person's desk, talking about a few lines that need improvement differs from laughing about a comic strip posted in a chat. It is also another way of criticism to have a regular appointment with collegues to talk 10-15min about a very simple pattern or technique. A last mode to consider is digging through the output of a static code analysis tool, playing the most neutral but complete blame game.

Now take the following lines and only look at them. Try not to judge or re-write them. Just let them find their way into your brain for about 2 minutes:

```
// is not within 6m radius
if (!isWithinRadius) {
} else {
// is within 6m radius
}
```

Have a look at a second example. Again try not to judge, only let it meander through your consciousness:

```
public class CompositumWithList {
  private List<?> listAttribute;
  // more attributes

  public boolean isFilled() {
    if (!attribute.isEmpty) {
      return true;
    } else {
      return false;
    }
  }
}
```

Now sit back for a while and find differences between the two snippets. It is obvious, they serve different purposes, that is not the point. Don't focus on the code but on the form, the meta level. We'll now walk to the discussion I came across, when I played Finding of the Week with the first example.

My immediate reply was, that I might be missing the true point but it looks like inverted logic. I often come across this and it only shows how the developer moved along the lines of a requirement. It might read like this: »As a dispatcher I want to get a notification if a vehicle leaves a 6m radius…« In turn the developer decided to make this the main condition of an if-statement. Maybe the else- branch contains some logging. There wasn't any more context. Also stepping back for a while and taking a second chance I didn't came up with something more elaborate. But the other developer said it is something totally different.

That is also why I posted the second example. Have a look at it, scroll up. There is more context and it is very isolated. Maybe even a non-Java-programmer immediately gets the idea of inverted logic. There's even a quick win for all, spotting the boolean return with the overcomplicated if-condition. With such an example I can split developers into at least three groups: those not responding at all (dark matter). Second there'll be some taking the quick win – absolutely correct but a little easy. Only very few will spot the inverted logic, focusing on the not-operator. This forces everybody to switch from regular to inverted logic and negate as well as double-negate statements to get the original idea.

After a little bit of opinion-ping-pong I outlined what I was thinking of as the better implementation:

```
public final class Radius implements Boundary {

  private final int rInMeter;

  public Radius(int size, Unit unit) {
    rInMeter=unit.translateInMeter(size);
  }

  // Inherited from Boundary
  @Override
  public boolean isInside(GeoPoint base, GeoPoint actual) {
    // heavy calculus here
    final int distanceInMeter=...
    return distance < radiusInMeter;
  }
}
```

This made my opponents point of view obvious. He not only responded, that this implementation violates the principle: keep it simple stupid (KISS) but he also labelled the comments from his snippet so called clutter – his example's main intention.

And this was also the tipping point in the discussion where it moved from neutral/ entertaining to personal/ insulting. From my point of view such a static pattern matching with KISS and clutter-comments is already covered by code analysis tools. And this is also the difference between the two examples, the meta level. Real humans need context and intention or a goal, at least some boundaries. Algorithms instead need clear rules. Static code analysis often works with regular expressions, pattern matching and a bit of statistics. They for example calculate the ration between lines of code and lines of comments. I don't think this is right and can be tricked in many ways. Nevertheless humans and algorithms agree: an entire program's code shouldn't contain the same amount of comments. But we differ, when it comes to single lines. There are Perl monks writing beautiful (short!) lines that need much more explanation why they work as they work.

My opponent in this game forgot, that humans write code but compilers and/ or virtual machines (VM) digest it. There is plenty of expert knowledge hidden in Java's compiler and VM turning my non-KISS Radius-class into native code that even gets optimized much further by the CPU. Executing it 10.000 or more times makes it as fast as the same hand carved C- or Assembly-code. (It does so indeed, what adds up is class management itself in Java, not creating objects or invoking their methods.)

But my example clearly expresses the intention: a circular fence with a radius fulfilling a basic contract of a boundary. It even goes further by translating units and also uses modifiers to inhibit extending this class or modifying the radius.

And regarding the so called clutter comments: I really appreciate them as often as they exist. This is rarely the case. But they allow archeology of knowledge – borrowed from Foucault. In the past there was a developer typing the comments regarding the 6m radius. Since there is nothing more visible in the snippet I cannot conclude, whether the boundary's shape is a square now. Maybe it is not 6m anymore. So I cannot label this clutter. From my point of view such a conclusion is short minded, drawn too fast and too superficial.

What I missed, right from the beginning, was to ask for the missing parts. I also took a detour to describe my intention. So my oponent felt degraded or I didn't value his past expertise enough – I didn't know him and neither did I know about his experience. So I was a bit superficial, too, assuming he didn't review much strange/ foreign code, yet. This was his tipping point and he took it (too) personal. I also must use both words, strange and foreign, because I chose the German word »fremd« back then, to bear exactly this idea: not only not his but also none of the collegue's/ company's code.

When I reviewd and audited code from other companies and even people from other cultures, I was thankful reading such *clutter*. Only these rarely used ingredients made it sometimes digestible. Most of the time it was simply garbage. The most entertaining Finding of the Week for me was, when an expert team from India mixed up the terms layer and shift. Their translation office turned the German requirements into English language. They ought to develop an application to assign workers to shifts. What they delivered was assigning employees to departments, layers, managers and such. It was very hard to read, because they used German class names like *SchichtController*. I could easily spot their fundamental mistake through a single comment reading *// this all doesn't make sense to me, but I had to deliver on time*. I still write emails with this developer today. We both don't work for the companies anymore. And we both learned more about people/ humans in general when coding than about coding itself.

So my recommendation to all reviewers out there is: mind the person behind the code. This goes beyond time and space. Not even Jesus can save all of us. And for the players of Finding of the Week: it is not about the laughter only, some take this fun serious. (And some of the participants might wrongly feel accused, possess a chainsaw and know where you live.)