Ugly Code #2

GREGOR WEGBERG – I Do Not Like Short Names

The Ugly

The last article ended with some quite wise words: "Never underestimate the power of meaningful names!" I think we have to talk about this a bit more. Take a short look at the ugly examples. It would not surprise me if you do not see any problem and this is an unpleasing reality for me. One huge challenge for me in Numerical Methods for CSE was the time which I had to spend to decode and understand the solutions. The same is true for many templates I got over the years here at ETH Zurich. Somehow, a lot of the assistants think of source code as a mathematical formulae and correspondingly try to use the same notations and names. This oftentimes ends up with very stupid variable names: "kkk" (this was for some code like this one: `k = k*3`), "ijk" (the multiplication of all three loop counters) or even "h" (a string containing some message for an exception). Makes sense – or does it?

This is an unpleasing reality for me

```
% Matlab: Ugly example 1
% variables given by template: a, b, func, n
% ...
for k = 1:n
  p = (a + b) / 2;
  xk = [xk, p];
  if func(a) * func(p) < 0
    b = p;
  else
    a = p;
end
% ...
```

```
// Java: Ugly example 2
t = new TaskWithTimeout("some.exe", timeout);
r = t.run();
if (r == ProcessResult.SUCCESS) {
  // ...
}
else {
  okay = false;
  if (r == ProcessResult.FAILURE) {
    // ...
  }
  else {
    // ...
  }
}
```

Example 1 is a very simple part of a Matlab solution. Because of its size and your education, you may even know what is happening here. However, is it because the code is self explanatory or because you know that it is part of a Bisection method? I am sure it is the latter. Variable names with one or two characters are non-self-explanatory. There are some exceptions such as "i", "j", and "k" for loop counters. Every programmer knows the meaning of them and uses them. It is a type of slang we use in our code, you would never find it in a dictionary, but everyone uses it nonetheless. However, I would even argue that those are worthy of a real name ("iteration", "iterCount", "run", …).

It is a type of slang we use in our code

Example 2 is another simple example taken from a Java method. Again, the example is simple and, because of this, you see after a short time what is happening here. However, this is again just because of the short size and your training in reading source code. I could change "t" and "r" to any other character, the source code would be still as explanatory as it was before. Sure, we could imagine that "t" stands for "Task" and "r" for "Result", but why not write those words right away instead of using some meaningless characters? ➔

The Beauty

A nice and short article around the topic of naming things right in source code files is referenced below.[1] It even goes a bit further and briefly discusses the problem of words like "Helper", "Manager", "Builder" and so on.

Example 2 was improved by incorporating our observation above. We use "task" instead of "t" and "taskResult" instead of "r". It may be wise to rename "r" into "result" if the variable is not necessarily containing a result of a task, but rather just a result of some computation. It is often tempting to name variables after the class or method name we call to get some value for the variable in question (i.e. instead of "t" use "taskWithTimeout"). I would plead that this would still be better than the name "t". However, this is a dangerous way of naming things. Variable names should reflect the content in a more abstract way. The name should not depend on the implementation but rather on what it represents. In the end you should be able to read the code like you would read a book and get an idea what is happening, what is used, etc. By renaming "t" into "task" we can use it with `TaskWithTimeout`, `Task`, `ExecuteVeryComplexComputationOnAllNodes`

A dangerous way of naming things

```
% Matlab: Nicer example 1
% variables given by template: a, b, func, n
% ...
for i=1:n
  midPoint = (a + b) / 2;
  intermediateResults = [intermediateResults, midPoint];

  if (func(a) * func(midPoint)) < 0
    b = midPoint;
  else
    a = midPoint;
  end
end
% ...
```

```
// Java: Nicer example 2
task = new TaskWithTimeout("some.exe", timeout);
taskResult = task.run();
if (taskResult == ProcessResult.SUCCESS) {
  // ...
}
else {
  okay = false;
  if (taskResult == ProcessResult.FAILURE) {
    // ...
  }
  else {
    // ...
  }
}
```

or any other task related class/method. This allows us to refactor our source code without renaming the variable because it represents the basic idea of what is going on.

In the nicer-looking Example 1, I basically gave every variable a name that represents their content better. For example, "p" is now "midPoint", which clearly expresses that it is a point in the middle of something. Of course, we could extend the name to reflect that it is in the middle of "a" and "b". However, I decided not to include this, as it should be evident from the context in this case.

Learn to type faster

I have talked to different people about this topic multiple times. The most frequently used argument for short identifiers, or even single-character names, is the fact that it is faster to type. Seriously?! Compared to, well, any other activity we do as programmers, the time we spend for writing around 10 characters more for each variable should be completely irrelevant. However, if it is a real problem to you: learn to type faster. This can be practiced! Plus, you save a lot of time down the line.

Reference

[1] http://blog.codinghorror.com/i-shall-call-it-somethingmanager/