Arlo Being Bloody Stupid

Conclusively demonstrating why you should listen to others…

Naming is a Process, part 4: Honest to Honest and Complete

August 26, 2015 by Arlo

In part 1 we talked about naming as a process. We talked about how legacy code is really defined by its poor legibility, and that reading is the core of coding. And we talked about how working effectively with legacy code is simply the process of having an insight, writing it down, and checking it in.

Later parts have gotten us to an Honest name.

Now let's look deeply at the third transition in names, from Honest to Honest and Complete.

I need to make my name fully describe everything this method does. Each level of improved naming should record more insights and make it easier for the next person to read the code.

This level actually makes it unnecessary for the next person to read the code. We want them to be able to trust that the name includes absolutely everything the method does so they don't need to read the body. They should be able to read and understand calling methods without having to come read the method we're fixing.

To attain this end we look through the body and find every single thing that it does. One thing at a time.

> **Where to look:** in the body of the thing you are naming.

The process is:

1. Find one thing that the code does (an effect, a calculation, a result, a state change).
2. See if that is already included in the name (it might be a legitimate substep of something that is already in there).
3. If not, add it.

Note: don't broaden the parts of the name. Your goal is not to find one abstraction that includes everything the method does. Your goal is to be precise; a reader should be able to read the method name and know *exactly* what the method does.

Imagine emailing the method name to someone. Just the name. Would they be able to generate exactly all of the things in the method body? They shouldn't want to add any more things and they shouldn't miss any.

> **Our insight:** one specific thing your thing does which is not yet explicitly stated in the name.

Ignore everything you've been taught about naming conventions. Long names are good. Conjunctions are good. Belaboring the point is good. Your only goal is to make sure that everything this thing does is represented in the name. And therefore anyone can trust the name. They no longer have to read the thing, just its name.

> **What we write down:** add a clause to the thing's name.

Really big things

Sometimes understanding a subcomponent requires extracting it and getting it to have a better name. This is especially common with long methods. When you do this, keep your focus on the main name. Extract a chunk and record insights only until you know whether or not it is important for naming your main thing. Then update the main name and move on.

Guard clauses are a good example. They're usually easy to identify but take up a bunch of space. I won't actually use them in the name of the thing (most of the time), but they can make it hard to read the rest.

I'll extract a bunch of what I think are guard clauses, get the name for the extracted component to be Honest and Complete, and see if they actually are all guard clauses. If so then I leave them alone and come back to the main method.

I'll extract guard clauses until I'm left with the residual. Then I'll analyze that more deeply and make sure I get everything into the name of my main method. I might inline the guard clauses again when I am done, or just leave them named something like _ValidateAllInputs().

In our running example I needed to take several steps to find all the things my method was doing. I eventually identified 4 main chunks of functionality, each of which happened in many steps. So my method became parseXmlAndStoreFlightToDatabaseAndLocalCacheAndBeginBackgroundProcessing().

Things that aren't methods

This step also applies to variable and class names.

- Classes should be named by all of their individual responsibilities.
- Variable names should include all the ways the variable is used.

In deep legacy code variables may be used in a surprising number of ways. I once had a poor, overused in/out parameter that I had to name searchHintThenBestMatchSoFarUntilItBecomesReturnValueOrReasonNoMatchWasFound.

Aside: naming by what it is or what it does

Here the insights that I'm having are all the important characteristics of the thing I am naming. I'm naming it by what it *does*, not by what it *is*. This shift is a critical step in eliminating god classes and long methods.

- When a thing is named by what it is, then it accumulates everything vaguely related to that identity.
- When it is named by every single thing it does, then our desire for shorter names drives us to have it do less stuff.

If you want a thing to collect functionality and grow, name it by what it is. If you want it to split and

shrink, name it by what it does.

This is the opposite of the Whole Value pattern. We use Whole Value when code has Primitive Obsession—when all the parts of what it does are scattered around the codebase and we want to collect them. Creating a thing and naming it by what it is causes programmers to naturally collect all those things. When we have a god class or a long method we want to split it. Naming it by what it does causes programmers to naturally split it up.

The Naming is a Process blog series

1. Good naming is a process, not a single step
2. Missing to Nonsense
3. Nonsense to Honest
4. Honest to Honest and Complete (this entry)
5. Honest and Complete to Does the Right Thing
6. Does the Right Thing to Intent
7. Intent to Domain Abstraction
8. Summary and Learning Path (will publish Tuesday, 9/1/2015)

Tags: design, legacy code, naming, naming is a process, refactoring, tdd

Simulator Agile pair programming working tiny learning example estimation naming architecture proficiency technical debt naming is a process legacy code measurement tdd refactoring feedback no mocks culture design

The last comments for
WET: When DRY Doesn't Apply

@jaybazuzi

For reference, here's James' talk from AATC 2016:

The last comments for
Naming is a Process, part 5: Honest and Complete to Does the Right Thing

EvilDBMethod

Hey, great article!
I really liked the practical approach to refactoring methods and using naming as...
» 3 weeks ago

The last comments for
Good naming is a process, not a single step

abelshee 50p

I'm glad you are finding it useful. Awareness of technical debt and how to work with it is the ...
» 3 weeks ago

The last comments for
Llewellyn Falco - What makes a good test suite?

kingroot apk new 14p

i thing granularity is the most importont among the 4 factors
» 4 weeks ago

The last comments for
Good naming is a process, not a single step

Filip

Oh man, this is just what I was looking for. What you describe as “indebted code” is something...
» 6 weeks ago

The last comments for
Naming is a Process, Part 7: Intent to Domain Abstraction

abelshee 50p

It will. But it is currently blocked on a software project. I mentioned the learning path? Well, I'm...
» 6 weeks ago

The last comments for
The Core 6 Refactorings

abelshee 50p

Interesting. Very different context / direction. I'm not looking for best.

I'm looking...
» 6 weeks ago

The last comments for
Naming is a Process, Part 6: Does the Right Thing to Intent

abelshee 50p

I hear you. I've heard others with the same request. I think it would make it better. And I'm...
» 6 weeks ago

Comments by IntenseDebate

Top 5 Posts

The No Mocks Book (33 comments)

Is Pair Programming for Me? (23 comments)

How I Learned to Stop Worrying and Love the Mock (20 comments)

Scaling Agile - the Easy Way (15 comments)

That's Not Agile (13 comments)

Comments by IntenseDebate

Theme: MistyLook by Sadish.