Arlo Being Bloody Stupid

Conclusively demonstrating why you should listen to others…

Naming is a Process, part 2: Missing to Nonsense

August 24, 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.

Now let's look deeply at the first transition in names, from Missing to Nonsense.

Sometimes we start with a reasonably-sized thing to understand (a class, method, variable, parameter, or field). Sometimes we don't. We have a large blob of code and have to understand it in pieces.

> **Where to look:** long things related to the task at hand. Long methods. Long classes. Long files. Long argument lists. Long expressions.

I look around for things that seem to go together. Examples include:

- A paragraph of statements
- A non-obvious expression (usually involving arithmetic or logic)
- A set of parameters that are frequently passed together.
- A set of parameters that are used together in the method (such as a bool which indicates whether another variable is valid)

I just pick one chunk that hangs together. It doesn't really matter how good a chunk I find. There are techniques that will find better chunks and speed up my understanding, but any chunk will get me one step closer. We can leave refinement for later. Good is too expensive; all I want is better (quickly).

> **Our insight:** one lump that hangs together.

We want to understand this lump of stuff. The first step is to create a thing to be named. Typical languages allow us to name any of 5 different things, so we must create one of these things. To do this, we use one of 5 refactorings:

- Extract Method (to name a bunch of statements)
- Introduce Variable, Parameter, or Field (to name an expression)
- Introduce Parameter Object (to name a set of parameters that are passed around together)

We don't care what we name the thing. A nonsense name is fine.

> **What we write down:** extract the lump to be a single, namable thing.

Check in or not?

Normally I check in after each step. But this time I might not. After all, we might not have actually made the code better. The name is nonsense. Previously, to read the code a person had to look at everything but it was all together. Now they still have to look at everything, plus realize the name is nonsense, and the code is no longer all together.

So in languages where I can do Missing to Nonsense quickly, I don't check in. I wait until I have Honest (which is also pretty fast).

However, many languages have crap tools. In this case, sometimes just splitting a thing out can take a lot of effort. Introduce Parameter Object in a dynamic language can take quite a bit of searching and careful updates. In that case I'll happily check in with a nonsense name. I need to save my game even if I went briefly backwards.

Aside: ways to find better chunks

Long methods tend to be organized as:

1. guard clauses
2. use parameters to read the data the method actually wants into local variables
3. process
4. calculate a result
5. either write it down somewhere or return it.

This means that the more important stuff is near the end of the method. So start looking for chunks from the bottom, not the top.

This also makes extraction easier in most languages, because functions often allow multiple parameters but only one return value. So it is easier to flow messy information (like a bunch of local variables) into a function than out. This is not a problem in languages with functions that have the same cardinality for both parameter lists and returns (support destructuring return like Python or support only one parameter like Haskell).

In general you're not reading code for the purpose of reading it. You're trying to accomplish something. Use that something to guide what code you bother reading.

Use binary search to quickly find chunks that are highly relevant. Look for large chunks that don't contain whatever it is you are seeking, or small chunks that do. Either quickly reduces the search space.

When you extract a large chunk of code you know to not be related, just get the name up to Honest and then move on.

Long methods are often dominated by a single control structure (after the guard clauses and fetching data into locals). The body of that control structure is often a good target. If there are multiple control structures in sequence, instead take the whole last control structure.

For example, if a method BecomeFroglike() contains one giant foreach loop, extract the body of the loop and call it MakeOne[whatever the control variable is]BecomeFroglike().

And if a method contains a foreach loop, then an if block, then another foreach loop, extract those into three methods, each taking the whole control structure. The outer method is then just a series of 3 method calls in sequence.

Exception handling is a little special. The body of a try block is often a good target for extraction. I usually name the resulting function *_Impl(). Complicated catch blocks are usually not worth extracting (there are exceptions), so instead look for ways to extract the useful bits away from them.

The Naming is a Process blog series

1. Good naming is a process, not a single step
2. Missing to Nonsense (this entry)
3. Nonsense to Honest
4. Honest to Honest and Complete
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

Tweet

Comments (2)

Comments (2)

Login

Sort by: Date **Rating** Last Activity

zainriz 0p · *47 weeks ago* +1

Interesting post. What does the Impl in *_Impl() stand for? Implement?

Reply **1 reply** · *active 47 weeks ago* Report

abelshee 50p · *47 weeks ago* +1

Implementation

Reply Report

Post a new comment

Enter text right here!

Comment as a Guest, or login:

Name	Email	Website (optional)
Displayed next to your comments.	*Not displayed publicly.*	*If you have a website, link to it here.*

Subscribe to None ▼

Submit Comment

- ## Categories

  Categories Select Category ▼

- ## Tags

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

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