# Arlo Being Bloody Stupid

## Conclusively demonstrating why you should listen to others…



# Naming is a Process, Part 6: Does the Right Thing to Intent

August 28, 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 a target that Does the Right Thing.

Now let's look deeply at the fifth transition in names, from Does the Right Thing to Intent.

Every improvement in our name until now has focused on what our class/method/variable does. We made it honest and then we made it do exactly what we want. The name is both correct and clear.

But it is awkward.

We want to use this name to understand calling methods. It doesn't yet serve that purpose. The name describes the thing. It doesn't tell us why we care.

That makes us stumble when reading the caller. We want the names to tell us the purpose of each subcomponent. But right now the name tells us what it does—implementation when we want behavior/intent.

Every time we read a caller, we have to check whether that implementation will give us our desired intent. That takes mental energy we could better spend elsewhere.

We've gotten as far as we can looking at our target to gain insights. Now we need to include the insights from its context.

> **Where to look:** at the calling methods / use sites.

Read through all the places where this thing is used. Understand it in the flow of other classes and method calls. What happens before, after, or with it?

Most likely the other names around this one will suck too. You need to improve them before you gain any insight. Sometimes insight will happen if the other parts are Honest, but you probably need to get to at least Honest and Complete.

When you see all the context items at the Honest and Complete level, you may realize that your target actually doesn't Do the Right Thing. This is rare but it does happen. Fix it.

As you start to examine the use contexts you will start to see a flow. The target is one step in a larger orchestration. It has a purpose. That purpose is usually at a higher level of abstraction.

> **Our insight:** the purpose for the code's existence. Why it might be used in a larger orchestration.

Rename the thing to state this purpose. Some common purposes are:

**Methods:**

- What it accomplishes (post-condition), regardless of starting state or process.
- Transformation it makes, regardless of beginning or ending states.
- Business process it replaces.
- Core responsibility the method takes on.
- What the method promises to allow callers to safely ignore.

**Classes:**

- The common responsibility of its methods.
- What the class is in the real world (Whole Value).
- The responsibility that calling context is delegating to the class and assuming the class with take care of. What the caller wants to ignore.

**Variables:**

- How this instance differs from other instances of the same type.
- The purpose the context has for this instance. How it is used.

> **What to write down:** a new name that is based on the purpose.

In our running example, storeFlightToDatabaseAndStartProcessing() becomes beginTrackingFlight(). The first name tells us what our method does. The second tells us why we care.

We don't need to know what tracking a flight entails. We just need to know that this method will take care of initiating that process for us.

We would expect to see a stopTrackingFlight() nearby. And perhaps some query methods to find information about flights that are being tracked.

## Play it again

Now repeat this step for another calling context. Your insight may be the same or your insightful name might be jarring in the other context. Often multiple callers are using the same code because of what it happens to do, not because of its purpose.

When this happens, create two methods that share an implementation. To the outside world they are different things because they express different intent. It just happens that to a computer they do the same thing…for now.

## Refactoring methods to share implementation

Do the following:

1. Extract method the entire body of the method.
2. Rename the extracted (inner) method to the name you had that Does the Right Thing.
3. Duplicate the outer method and edit its name to something guaranteed unique (nonsense is good here).
4. Rename the nonsense method to the new purpose.
5. Edit the call site to use the method with the new purpose.

The reason we hand-edit to nonsense and then rename is so that our tool can catch errors. Hand-editing code is dangerous and something I tend to avoid.

I could duplicate a name or create aliasing problems, especially in the presence of inheritance or global functions with imported namespaces. But these are all problems that the rename refactoring checks for. By hand-editing to unique nonsense I reduce the chance of making a mistake. And then renaming to the final name ensures that I don't introduce one at that point.

## Refactoring classes to share implementation

For a class the recipe depends on how your IDE does Split Class.

If Split Class supports delegation:

1. Split Class and select everything. Choose the option to have all callers use the old type and create delegating members.
2. Rename the inner type to the name you had after Doing the Right Thing.
3. Copy the file for the outer type (assuming class-per-file).
4. Hand-change the class name & constructors in the copied file to something guaranteed unique. Nonsense is good here.
5. Rename the copied class to the new purpose.
6. Update the construction site to instantiate the new class.
7. Follow the compile errors, applying the auto-fix each time to change the type of the variable.

If Split Class doesn't support delegation:

1. Create a new class with a unique nonsense name, a single private field of the old type and constructors that match the old type. Use auto-generate members to create delegating members.
2. Rename the old class back to its Does The Right Thing name to make sure that doesn't cause conflicts.
3. Undo the rename refactoring (it was just a test).
4. Hand-edit the name of the outer class to the name the inner class has (the Intent name for the other context).
5. Hand-edit the inner class to the Does the Right Thing name.
6. Update the type and construction of the field in the outer class to use the correct type.
7. At this point, all call sites will now be using the outer class via the purpose-based name.
8. Pick up from step 3 in the recipe that assumed Split Class that supports delegation.

This recipe uses a common mini-recipe. We want to introduce a new name for an extracted thing but leave all users using the outer thing.

If we can't extract the inner thing, then we can get the same result by creating a new outer thing as a shim and then hand-editing the names (not updating call site) to slip it into place.

It also uses a common refactoring test strategy. The most useful part of refactorings is not that they change your code, it is that they tell you what code changes are safe. Sometimes a refactoring won't do exactly what you want, but you can still use its analysis to verify that the thing you want to do is safe.

We do and undo a rename refactoring to test whether our new names will create any conflicts, even though we execute the name changes by hand so that we can put the shim into place.

## Warning: you can get nonsense

This step is the easiest one to screw up. If you screw it up you will get a nonsense name. You won't notice; the name will make sense to you right now because you have full context. But it will be nonsense the next time you try to use it.

The problem is that this step is intentionally information-losing. We are replacing some of the information about what our thing does with information about why you care to use the thing. We are asking the potential users to trust us. And we have to earn that trust. We earn it by using consistently clear names and code that Does the Right Things.

## Ways to get nonsense

An extremely popular way to get nonsense is to name a thing by when it is used, rather than why it is used. This gets us right back to calling something preFetch(). One variation on this is to name a method by its initial condition. Ignore initial conditions in method names; they always end in broken abstractions.

Another popular route to nonsense is to create a new concept that is never used anywhere else. One-off abstractions are noise. Create your abstraction by looking at many named entities and use it uniformly.

But perhaps the most popular route to nonsense is to use a CS-y term. The domain of software engineering does not belong in your names unless you are writing a programming language. Even then, domain-specific languages are easier to read.

To avoid nonsense be specific about the context. Clearly state why someone would want to do the things that this class/method is doing. Keep any parts of the Complete and Honest name that make it more obvious when you want to use this thing. Drop the rest.

## 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
5. Honest and Complete to Does the Right Thing
6. Does the Right Thing to Intent (this entry)
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



Comments (6)

Comments (6)

@ifyouseewendy · *8 weeks ago* 0

Hi Arlo,

I'm digging about this "Name Driven Design" topic. Your post is really brilliant. I learned a lot from it. Thanks.

When reading this post, I think I can understand the basic idea here. But it's a little hard to follow the steps without any specific examples. If there was a repo, each commit doing one step, it'll be wonderful. Or maybe you can just leave a gist about the original version, everyone read this post can practice the naming and refactoring process.

Any way, thank you for sharing this wonderful post.

Reply **1 reply** · *active 6 weeks ago* Report

abelshee 50p · *6 weeks ago* +1

I hear you. I've heard others with the same request. I think it would make it better. And I'm likely to get to it at some point, but not immediately. I had the time and energy to get the first solid pass out & refine it a couple iterations, but not yet the time to create code samples for each step (there are samples for a couple, but not most). I would be happy to link to good samples that someone else created. Or I'll eventually get to samples of my own. But that'll happen after I get the last one in the series.

Reply Report

aliraza · *11 weeks ago* 0

Great post.

Reply Report

Jay Bazuzi · *46 weeks ago* 0

Up until this point, the process was almost mechanical. The code tells you the structure it wants to have. But the step to Intent is risky. I hesitate here, and I think that's a good thing.

Reply Report

EricGu · *46 weeks ago* 0

Arlo,

I'm really enjoying this, but I think it would work better with more examples. In "ways to get nonsense", you talk about some pitfalls but they would resonate more with examples.

Reply **1 reply** · *active 41 weeks ago* Report

abelshee 50p · *41 weeks ago* +1

Yeah, I see what you mean. I'll see if I can go back and add gists & the like at some point.

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**

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

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