Arlo Being Bloody Stupid

Conclusively demonstrating why you should listen to others…

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

August 27, 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 and Complete name.

Now let's look deeply at the fourth transition in names, from Honest and Complete to Does the Right Thing.

In making the name complete, we have now made it possible to reason over the responsibilities of the class/method/variable without having to read its full implementation. This sets us up to change its responsibilities.

At this step I look only at the name of the thing I'm working with. I Ignore both its usage sites and its body. Does this name make sense?

There are typically 2 kinds of clauses we want to eliminate. Each is found by one key question:

- Is this clause unrelated to other clauses in the name?
- Is this clause a concern we want to encapsulate?

> **Where to look:** at the name.

Again we just have an insight, write it down, and look for another.

> **Our insight:** one clause we don't want to have this thing expose.

Writing it down requires structural refactoring. We need to keep the name Honest and Complete. If we don't like the name then we have to change what the thing does so that we can use a different name.

What we write down: extract or encapsulate one behavior of the thing.

This usually involves one more refactoring:

- Inline

Safely moving a responsibility from a method

When I want to split a responsibility out of a method, I often do the following sequence:

1. Extract method all the stuff before the part I want to break out.
2. Extract method the part I want to break out.
3. Extract method all the stuff after the part I want to break out.
4. Split the name of the outer method up into names for the 3 parts based on what does what chunks.
5. Add any missing clauses required to make the names complete. For example, the above split may divide a calculation from the part that writes it somewhere. The outer method may just be named … AndRecordYield…(). So I now have one inner part named …AndCalculateYield…() and another named …AndRecordYieldToService(). The methods are smaller so it becomes more obvious how to be specific in their names.
6. Inline Method the outer method. Now all call sites use the 3 methods directly and you have split out the functionality.

Encapsulation

Another common case is to want to encapsulate something. This usually entails not just removing a clause from the name, but actually encapsulating the behavior so callers can't see the effect. The problem is typically that the behavior is performed on some parameter.

An example is the handling of the local cache in parseXmlAndStoreFlightToDatabaseAndLocalCacheAndBeginBackgroundProcessing(). I'd like to encapsulate that cache and use it as a read-through cache just for performance.

The solution is to find some sequence of refactorings to shift that parameter to be a field. Sometimes this involves creating a new class, splitting a class, or moving the method we are working with to a different class. It also usually involves changing object lifetime and removing references to the value from all callers of the method, often several layers up the stack.

The easiest way to do this is typically to refactor until the method uses everything via a field instead of a parameter, then use the auto-fix to remove unused parameter, then go up the call-stacks and continue removing unused parameter as far as you can.

Here is one common sequence, used when we start with a static method.

1. Use Introduce Parameter Object. Select just the one parameter you want to encapsulate. Name the class Foo and the parameter self.
2. Use Convert To Instance Method on the static. Select the parameter you just introduced.
3. Improve the class name (Foo) to at least the Honest level.
4. Go to the caller of the method. Select the creation of the new type. Introduce parameter to push it up to the caller's caller.
5. Convert any other uses of the parameter you are encapsulating to use the field off the new class.
6. When the last usage is gone, use the Autofix for remove unused parameter to remove the now-encapsulated field.
7. Select any usage of the public field in the calling method and Extract Method the related statements / expression.

8. Convert to Static on the extracted method, then Convert to Instance Method to move it to the new type.
9. Repeat 7 & 8 until the calling method no longer uses the field we are encapsulating.
10. Walk up the stack to the caller of this method. Repeat from step 4. Stop when you get to the initial fetch / creation of the value you are encapsulating.
11. Move that fetch / creation to be a factory method on the new type (via Extract, Make Static, Convert to Instance).
12. Repeat from step 4 for any other callers of your original method.
13. At this point the only references to the value you are encapsulating will be within your new class. The only users of the constructor that takes that value will be factory methods / other constructors.
14. Convert the constructor to private.
15. Inline the public property to access the value you are encapsulating. Now all the class methods will just use the field.

The recipe is long, but each step takes 1-2 seconds with proper tooling. No step requires editing code. You can encapsulate even a highly-used value in 2-10 minutes with practice.

You can also encapsulate multiple related values at once. And a variation can be used when you want to split a class or handle similar cases.

Classes and variables

Refactoring a class to Do the Right Thing usually means executing the Split Class refactoring one or more times. Each split pulls out one responsibility.

Refactoring a variable usually means just introducing a new local variable and changing which one is used when in the method.

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 (this entry)
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

Comments (1)

Comment (1)

Sort by: Date **Rating** Last Activity

EvilDBMethod · *3 weeks ago* 0

Hey, great article!
I really liked the practical approach to refactoring methods and using naming as a methodology for deciding what and how to refactor. However I think one small thing can make this article even more awesome - you are mentioning different code refactoring

techniques such as "Extract Method, Inlining" etc but no links or explanations about them. I'm mostly familiar with them from Martin Fowler's blog/books and it would be great if you could add some reference or link to an outside (or inside if you wrote about them) source explaining this methods.

Thanks again for the article!

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

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.