THOUGHTFUL CODE

Code is Poetry

HIGH-LEVEL CONCEPTS

Rubber Duck Debugging: The Psychology of How it Works

JANUARY 9, 2019 | DAVID | 8 COMMENTS

Computers process information differently than humans do. Anyone who's first learning to program understands this well. What's hard about programming for a beginner isn't really big hard esoteric concepts, but that you've got to be so painfully exacting in how you describe everything to a (dumb) computer. That's why we do rubber duck debugging.

But to keep on my first thread, because it illustrates why rubber ducks (or inanimate objects, or pets, or speechless-children) can help you fix errors in code. After we explain that a bit more, we'll get to making sure we know why "rubber duck debugging" is such a popular technique, and what that tells us about human psychology.

The Exactness of Computer (and Duck) Understanding

It looks cold here to this human

“Is it cold outside?” is a question that most humans, having some idea of the weather, will answer pretty easily. They’ll say something like, “No, it’s pretty nice.” Asked that question, a computer — or a really finicky and hyper-rational person — will need you to define each of those words. To quickly illustrate, “cold” is the fuzziest of the bunch and can be answered relative to many standards. A few:

- Absolute zero is cold, so no temperature on the surface of planet Earth can possibly be described accurately as cold.
- On a summer evening in a temperate climate, most humans would describe 48ºF (9ºC) as cold. So the current temperature being 42ºF is clearly “cold.”

- On a winter day in a temperate climate, most humans would generally say that 42ºF is not really “cold.” 20ºF (-6ºC) is “cold.”

Without having one of those standards picked for it, most computers would have no idea how to answer the question of it being “cold.” Even if it knew with a high degree of certainty and precision that its was 52.9ºF outside, with a mild breeze and few clouds.

Why explain this so clearly. Because it drives home the point that most human logic is “fuzzy” and most computer logic is not. We need to use plastic-y duck-shaped toys when programming because human fuzziness is an issue for computers.

Why should you tell a duck about your programming problem?

Quite simply, we use ducks and other stand-in because of the vast difference between the way most humans think and the way that computers do. When you hit a roadblock when programming, it’s likely that you’re not thinking the way a computer does. Computers are painfully precise. Humans are, generally, forgiving and loose in their understanding. So many of the “bugs” and issues you have when solving a problem in your program originate in not being clear and explicit enough in your instructions.

This is where the concept of "rubber duck debugging" comes in. To be clear, the rubber-y and duck-y-ness of the object don't matter, it's just important that it's not you. If you can imagine a figure, that works. Your dog or cat works. Even your pet grasshopper. Or your office chair, your teddy bear, a bottle of hot sauce, or even a coworker. (Notice that I intentionally left coworker for last. The insight of another programmer isn't part of this exercise, and you can save them the hassle by trying any other object first.)

The important thing is that you explain your problem to this object of (one-sided) conversation. Follow the steps:

1. Explain your (broken) code and its goals, generally. Don't worry about details, just set the context for your duck.
2. Line-by-line, explain what the flow of the whole function or method that's not working is. Don't skip details, ducks love details!
3. If your duck friend hasn't spotted it yet, be sure to explain all the intermediate states and transitions in detail.
4. **Find your (stupid, obvious) solution!** The reason we so love "rubber duck debugging" is that (typically) in the process of doing this, you'll find your problem. You made a small typo. You were operating on the wrong variable. Your `if` condition was checking the opposite of what you thought. Something is likely to be revealed.

The Psychology of Rubber Duck Debugging's Effectiveness: Two Shifts

If you’re like most, when you’re explaining something to someone else two important things shift in your head. **First, you’re likely to slow down and be more exacting than you are when you’re power-typing code.** Most of us think way faster than we talk. So especially if you’re verbally explaining what’s going on to this other object, you’re likely to be a bit more careful and precise by virtue of that speed bump of saying it.

This effect is linked but distinct from **the second shift, which is that you have to work from the knowledge that the rubber duck you're explaining your code to doesn't know as much about the problem as you do.** It's a milestone (of the concrete operations stage) in Piaget's theory of cognitive development when a child learns that other people might have different understandings than they do. For anyone reading this, you almost certainly have this skill and don't even realize you do. And it's what's making rubber duck debugging effective for you.

"Teaching" Your Rubber Duck to Code

When you're assuming the ignorance of your rubber duck, you're having to explain more thoroughly and exactly than you were likely thinking those specific lines of code through in your head. You're forced, by the need to be precise while helping someone else understand your problem, to pay very careful attention to all that you were previously just taking for granted.

If you've possibly heard people recommend teaching as a great way to further your learning, it's rooted in the very same shift.

When you’re explaining, “and then this probably will happen because it usually does” feels pretty lame as an explanation. So you’re forced to understand more deeply and explain more fully.

It is precisely this that makes rubber duck debugging effective. The psychological shift outside of yourself, and into the rubber ducky, changes everything. Suddenly your perspective is different. Suddenly you think differently than you were in that rut of brokenness. And that’s super important.

Onward, to bigger problems! Beyond the rubber duck

By explaining your program to a duck, you often magically find the flaw in your code. As an answer on StackExchange points out, where this technique really shines is when you’ve got a simple logical flaw that no amount of variable state checking will reveal to you. Because of the psychology of how we humans relate, you’re forced to think fresh when you invoke your little rubber ducky, and your problem may be quite simply solved. I hope it works for you!

8 thoughts on "Rubber Duck Debugging: The Psychology of How it Works"

Pingback: Study Smarter, Not Harder – Emma McAleavy

MOHAMMAD DAUD IBRAHIM *says:*

Stepping outside of yourself and reflecting on your code from a different perspective helps in revealing our own flawed and distorted assumptions and understanding.

AUGUST 11, 2019 AT 11:34 AM | REPLY

DAVID★ *says:*

Yes. And as I think you're suggesting, that applies outside of programming too

AUGUST 12, 2019 AT 1:55 PM | REPLY

NICK HODGES *says:*

I also find that writing out a question for StackOverflow will help me find an answer before I actually submit the question.

AUGUST 11, 2019 AT 6:24 PM | REPLY

DAVID★ *says:*

Yes! Same basic mechanism at play. Anything that helps you step out of your present vision of the problem is big

AUGUST 12, 2019 AT 1:54 PM | REPLY

MITCHELL J. SANDERS *says:*

I'm a software engineer professionally, but have been working on a sci-fi/fantasy novel at nights. I actually found that rubber duck debugging helped me get through some writer's block during the outline process. Who would have thought that technique could apply in other domains? Great post.

APRIL 12, 2020 AT 6:26 PM | REPLY

CHRIS *says:*

This is very a similar approach to what I’ve called the ‘cardboard cut-out programmer’ effect. When you ask one of your colleagues to help ‘cos you can’t find the problem and it’s a few minutes before they’re free to help, you start thinking what they would do and what questions they would ask. And 9 times out of 10 you find the answer yourself.
So you don’t need a colleague, all you need is a cardboard cut-out of them looking over your shoulder and think ‘what would they ask’.
Having said that, not as cute as a rubber duck…

DECEMBER 8, 2021 AT 3:19 AM | REPLY

OGHENEMERU G. AVWEMOYA *says:*

This!!! Some months ago, I was working on a project and hit a roadblock, after days of debugging without making progress, I got in a chat with a senior colleague. I started typing everything about what I was trying to do and where the suspected issue was arising from. He wasn’t online to respond me when I sent the breakdown so after some minutes, I read through what I explained to him and immediately, I discovered where the “moth” was in my code and that was the height of the debugging for me. Immediately I rectified it, it worked as supposed and I sent him a message that typing the problem in his DM’s was the debugging I needed. :)) Great read there David!

MARCH 15, 2022 AT 12:39 PM | REPLY

Leave a Reply

Your email address will not be published. Required fields are marked *

COMMENT *

NAME *

EMAIL *

WEBSITE

POST COMMENT