|  |  |
| --- | --- |
| "Essentially, the important thing is that you explain your problem to this rubber duck or baby. Explain your goals. Line-by-line explain what the flow of the whole function or method that's not working is. Explain all the intermediate states. And (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**When you're explaining something to someone else, two important things shift in your head. First, you're likely to slow  | "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 your 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 that it's not you. It can be your dog, your cat, your baby, your office chair, your teddy bear, a bottle of hot sauce, or even a coworker. (Notice that I intentionally let coworker for last — the insight of another programming isn't part of this exercise, and you can save them the hassle by trying it on any other object first.).  |
| "down and be more exacting than you are when you're really 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 just by virtue of that speed bump of saying it all."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.  | **The Psychology Underlying the Power of Rubber Duck Debugging**By David Hayes https://pressupinc.com/blog/2014/06/psychology-underlying-power-rubber-duck-debugging/**So why does that mean you should tell a duck about your programming problem?** "Quite simply, because of the vast difference between the way most humans talk and 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.  |
| "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."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. 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  | TEACH ME KNOWLEDGERubber Duck *Debugging*  |
| "as an explanation. So you're forced to understand more deeply and explain more fully."And magically, a lot of the time you find your flaw in that exercise. 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". | Learning ObjectsDecember 2016Mita Williamsaedileworks.comcc-by |