Rationally Speaking #233: Clive Thompson on “The culture of coding, and how it’s changing the world”

Julia: Welcome to Rationally Speaking, the podcast where we explore the borderlands between reason and nonsense. I'm your host, Julia Galef, and my guest today is Clive Thompson.

Clive is a journalist covering technology and culture. You may have seen his writing for Wired, among other places, and he's the author of two books. Including most recently, Coders: The Making of a New Tribe and the Remaking of the World, which is a fun and interesting intro to the psychology of coding and the people who love it, and how that psychology and that culture ends up influencing what they make and their impact on society.

Clive, welcome to Rationally Speaking.

Clive: It's good to be here.

Julia: I don't know if you remember this, but I have to tell the story of how I ended up having this podcast with you. A few months ago, you wrote an article for, I think it was The New York Times-

Clive: New York Times Magazine, yep.

Julia: Yeah, New York Times Magazine, and in it you mentioned some study, and your article got shared widely on Twitter. Then a friend of mine, Kelsey Piper -- who is also a recent guest on the podcast and is a journalist at Future Perfect -- she responded to that thread saying, "Actually, the study in this article is terrible, and shouldn't be cited," and then her tweet went viral criticizing the study and your article.

You replied very graciously, and said, "This is a good critique of the study, I'm going to issue a retraction or revision, and I'm also going to revise the part of my book that talks about this study."

I was so pleased at your response that I instantly bought your book and encouraged other people to buy your book too, for two reasons. One, because I have more trust in and more interest in the things that someone has to say if I know they're the kind of person who corrects misinformation when they find it in their writing. Two, because I want to incentivize that kind of behavior, because it's good for the epistemic commons. So, thank you for that.

Clive: Not at all. I mean honestly, I was very thankful to Kelsey for speaking up, because I mean, you know when you're a magazine writer, there's a fact-checking process for the magazine, but everyone knows that sometimes

things get through. You're trying as hard as you can to get things right, if you make a mistake, you really want to correct it quickly, particularly if it's for a magazine where it's going to be on the web for as long as the internet endures, or the sun explodes, or something like that. Getting it right is good, and I'm very, very grateful to people that pick things up and let me know.

Julia: Although, I have to say, I ordered the book, and then it's been months since I ordered it. And I had it on my shelf and I forgot that I had ordered it for this reason, and I guess I thought that the publishing company had sent me a review copy or something. When I started reading it I was like, "Oh, I should interview Clive," and I had no memory that this was how I ended up with your book. So my decision to interview you was sort of independent of this epistemic virtue reason.

Clive: It came to you twice.

Julia: Exactly, two reasons. Anyway, Coders -- one of the things I liked most about the book is the way you really delve into the psychology of coders and how that affects society. Probably your most emphatic generalization that you were willing to make about coders was their love of optimizing and efficiency, which really resonated with me personally, actually. Can you start to explain what's behind that love?

Clive: Sure, sure. Well, so one of the things, the reason why this cropped up over and over again when I talked to developers is because when you're learning to program a computer for the first time, one of the things you quickly discover is that computers are really good at doing the things that humans are really bad at, and vice versa.

Humans are, you know, we're great at intuition and synthesizing in a very organic way different pieces of information, but we are terrible at being meticulous. I mean, if you ask us to do something repetitively over and over again, we drift, we space out, we get it wrong. We're terrible at doing things at the precise time we're told to do them.

In contrast, you know, computers are amazing at doing things repetitively over and over again with great precision, in precise timing.

One of the things that dawns on you when you're learning to program is that, "Wow, there's all these repetitive tasks that I have to do in daily life that actually could be better done by a piece of software." You know, like you've got some job that requires you to take all these different fields from a Word document and dump them into a spreadsheet, and it normally takes you four hours to do that. You're like, "This is an insulting waste of my time."

And when you learn a bit of programming you're like, "Wow, I can actually write a little routine that will do that in a matter of milliseconds, and it'll do that every time critically for the rest of my life. And actually I can give that to all my colleagues, and I can suddenly save, not just me, but the entire corporation, hundreds of hours in a year."

You discover that, and you start automating things in your everyday life, and it becomes very addictive. There's something incredibly delightful about optimizing all these dull, boring tasks. And it starts to become almost, the way coders have described it to me, almost like an aesthetic. Like inefficiency and un-optimized repetitive behavior sort of grosses them out, you know, in the way I think someone else would be appalled by a bad smell.

Julia: Yeah, I remember this metaphor that you mentioned in your book -- it wasn't your metaphor, you were talking about how coders talk about code. You said that when code is efficient, they talk about it as sort of “clean, beautiful, elegant,” kind of like a visual art metaphor.

Then when it's a terribly-written mess, the metaphor is that it “smells bad.”

You had a nice analysis of what's underlying that metaphor, do you remember it?

Clive: Yeah. They shift from the pleasures of the eye, the apprehension of gorgeous proportion in a painting, you know, "Wow, that code is beautiful," to like, "Oh my God, that code smells." It's like there's something wrong with it, and it's just like someone put a fish in a paper bag beneath the floorboards and it's rotting. There's these very funny kind of aspect to the way coders think about what they do, and the metaphors that come up.

Yeah, the efficiency thing is really interesting. I heard it, I wouldn't say universally, but very, very commonly from a great array of coders I spoke to, all walks of life.

Julia: I was just thinking about that metaphor. Because I love metaphors that kind of work on multiple levels, or reveal something deep about the thing they're metaphorizing. The reason that the bad smell, like "this code smells bad" seems like such a good metaphor to me, is that when you smell something bad, you don't know where the smell is coming from.

Clive: That's right.

Julia: It's like, "Somewhere in this code is a problem." But also, it often indicates something kind of growing, or out of your control, not planned. Like

something rotting, or a colony of insects or bacteria or something like that. And it's not ordered, and it can be contagious. Like, if you don't take care of it, this bad code could mess other things up, or the problem could spread, or something like that.

Clive: Right, and that is exactly what – like, when code's a real mess and there's something wrong with it, that's exactly the reason why the metaphor I think is so powerful. Because those are exactly the emotions that the coder faces. They're sort of looking at this thinking, "Oh God, this is like thousands of lines of spaghetti code, and it smells terrible. There's nothing going wrong right now, but it could so easily go wrong because I can't even understand how this even works, it was so badly written. I mean, no one documented any parts of it."

You're just sort of aware that there's something terrible out there, and you could dive in and clean it up, but like you might not have time to do that. You might have to tolerate this horrible smell, but it's quite aggravating. I think that's why the smell metaphor works so well.

Julia: Yeah, it's so perfect. You started learning to code in the process of writing this book, right? Was that your first foray?

Clive: Yeah, I mean I'm of the generation where I did a little bit of coding when I was a kid in the 1980's. On the first generation of personal computers, like those ones you could plug into your TV, like the Commodore 64 or the TRS-80.

Honestly, I loved it. I found it just incredibly joyful and fun, and I made databases and I made games and I made chat bots. I probably, I might've even become a programmer, except my mother ... My father was a civil engineer, and he was totally into technology, but my mother was worried that I would just sit around playing games all the time, and I would drop out of school. She said, "Yeah no, there's no computer in this household."

We never got one. And I would write code literally on pieces of paper, and try and type it into the school computer when I could get access. You know, if you don't really have a machine, you can't really go very far. I sort of decided to leave it behind, and I decided to become a writer instead, but I never really lost the fascination of it. Which is sort of why, the instant I started writing for a living, I immediately gravitated towards writing about technology, and writing about its effect on society.

Then, you know about seven or eight years ago, I guess five or seven years ago, I decided I wanted to learn some of the new languages that were common, like Python and JavaScript. I was poking around on them and teaching myself stuff, and that sort of was around the time that I started thinking about writing this book. I also wanted to be able to talk more

confidently with the people I'm interviewing, so I wanted to do enough coding that I could really apprehend what was hard and challenging and amazing about the work they were doing, as sort of a useful reporting exercise.

It turns out that, again, I discovered that I really loved it. In fact, I think I liked it more than writing. I would procrastinate on writing my book by coding, basically.

Julia: Nice. Is your mom kicking herself that she pushed you out of what would have been a very lucrative career in tech, and into a career in journalism?

Clive: You know, I think she notes the irony a little bit, for sure -- but I mean, the truth is I'm happy being a writer. I think it worked out well for me. I think if I had hated my writing life, I would have an even more troubling relationship to not being a coder. I'm kind of getting the best of both worlds, because I get to have fun doing this writing stuff, and I get to sort of have this weird little pallette that I use in the service of my journalism.

I mean, like one of the things that I discovered that's kind of interesting that I think might be resonant for your listeners is, I think there's actually a certain incredible usefulness in knowing a bit of coding when you're not a full time programmer. Like when it's an adjunct power onto something that you're doing -- whether you're a nurse, or you're in marketing, or you're in journalism.

Because you can do this like crazy little automation thing. Like when my book came out -- you're going to find this funny – so, my book comes out like a month ago, and what does every new author do for the first few days? Well, we sit in front of our Amazon page refreshing it over and over again, to see whether or not the sales rank is rising at all. You know, it's like the totally neurotic "does anyone love me" thing.

After doing this for a couple of days, I decided, "Okay, this is crazy. A, it's not psychologically healthy, B, this is a repetitive, computer behavior and I can automate it." I sat down and I wrote a little web scraper that goes into that page four times a day, retrieves exactly the information I'm looking for, and formats it into text messages and texts me. I have a little automated bot that does that on my sales rank for me.

Julia: Well -- first of all, kudos on the coding in the wild. Is it better for your psychology than the clicking, or is it just that you're no longer tethered to your computer?

Clive: Yes, actually it is. It's broken my habit, I swear to God. Like once I had this up and running, I just stopped going to the page. It's kind of nice, like I

actually keep track of generally how it's doing, but I don't have to think about it, and I've stopped thinking about it.

I think in some respects, it was almost like a form of cognitive behavioral therapy, writing the code, because in the process of writing it, I'm like, "Okay, I'm insane, I'm insane, I'm insane." You ponder deeply your insanity as you turn it into an abstracted algorithm, and then you dispatch it to let this deathless computer execute it on a schedule.

Julia: You talk in the book a little bit about the downsides of this love of optimizing or efficiency. But the examples that I remember, and I might be missing something or misremembering, were things that felt kind of like obvious mistakes. Like, there was that dude who was obsessed with reducing the amount of time that people spent making jokes in meetings. And there was the guy who invented an app that would automatically send love messages to your partner.

Those are both obviously dumb mistakes, right? Do you see a problem with --

Clive: There are also bigger ones, though. Like, think about the like button, okay? Originally, the engineers and designers who made the like button conceived of it entirely as an optimization, as an efficiency ploy. They were noticing the fact that before the like button was there on Facebook, it was actually kind of inefficient and a little ponderous to indicate that you liked someone's photo. You had to write a comment, you had to go, "Hey, great photo, Clive." The truth is, most people are just busy and rushing and they're not going to do that.

The designers thought, "Well, you know, we could probably unlock a lot of positive behavior if we made it one click, easy to approve of a post, just dramatically more efficient, to indicate your approval." They get together -- Leia Pullman, the designer, and Justin Rosenstein, the programmer -- and they prototype it, and they got it working. They originally called it the "awesome" button, you were just going to click on something and say that it was awesome. Anyway, they put it together and they showed it to Mark Zuckerberg and the top team, and eventually they said, "Yes, this is a great idea, we'll do this."

They roll it out -- and you know, the first order effect is exactly what they expected. There is, as they said, an "unlocking of latent positivity". When you make something easier to do, more efficient to do, people will do more of it.

But over the next year or two, they began to sort of notice rather uneasily that they also inadvertently created a bunch of kind of unsettling behaviors, that they didn't particularly like, and wished didn't exist. For

example, people started posting something and then sitting there, much like me with my refreshing of my ranking page on Amazon, just constantly refreshing the page, to see whether they were getting any likes. Once you quantify something... This is Campbell's Law, if you create a metric, people will alter their behavior to boost the metric.

They realized that what they'd done is they'd created a situation where people were now sort of hustling for likes, and deforming what they posted in the first place, with the idea that it was specifically to try and get more likes.

In some respects, they actually really regret these knock-on effects that they did not expect were going to emerge. Because they successfully did what they set out to do, but they created a situation that they think actually kind of ruined aspects of Facebook. In fact, actually, here's the thing, think about it this way: every time there's a major collision between a large software company and the rest of the world, it's nearly always because they dramatically increased the metabolism of something in a way that broke other things. There's no pure winners in efficiency. Every time you increase something, increase its throughput, you're going to cause some sort of outcast effect.

Uber is a wonderful optimization ploy. The company identified a serious inefficiency in the way that cars are dispatched -- people don't know where the cars are, cars don't know where the people are. They were like, "That's crazy, like we have these pocket computers and we can resolve that inefficiency," and did an amazing job at it. They produced an enormous win for me, the rider. It is crazily easier for me to get around now using cars.

They also created kind of a cool, new opportunity, because now it became a lot easier to become a driver. You didn't have to go through this process of finding someone's cab license in this completely oligopolistic, slightly mobbed-up world of cab licenses and cab medallions -- you could now just download the app and become a driver. If you wanted to earn a little bit of extra money, enormous upsides.

But they also kind of along the way broke the ability of a lot of people to make a full time living, by distributing the work amongst a lot of part time gig people. This route to the middle class that had existed for decades -- it was not a great route, like it wasn't like anyone thought it was a fantastic way to make a living, but it existed, particularly for immigrants. It's now more or less gone, right? That whole thing, that was part of the civic life of cities, is broken.

Julia: Okay, so those seem like two very different types of optimizing and effects of optimizing. The first one, the like button, seems like a genuine kind of

unintended consequence. Which -- I mean, I guess it wasn't even quite making it easier to leave a comment. It was like creating a whole new way of interacting with the post, that had a different meaning. It was like voting for the post, or something, as opposed to interacting with the person. But that still is kind of an interesting, unintended, if not negative, then at least “double-edged sword” consequence of optimizing.

But the Uber case just seems like purely, straightforwardly, this is what innovation is supposed to do. It's supposed to find new and better ways to do things. And there are always losers in that process. You could totally make an argument that society should have a cushion for the people who get displaced by new technology, and so on -- but that doesn't seem like an argument against optimizing or efficiency.

Clive: Well, I mean, I'm not sure that I'm arguing against it entirely. I'm merely pointing out that this is what the central trick of software is. I mean, over and over again, if you ask, "What is the practically largest pattern of what software does in society?"-- It speeds things up. It gets rid of inefficiencies, by and large.

You know, like Microsoft Word basically took the process of creating a document -- which I'm old enough to have done on a typewriter, and it was ruinously slow -- and it made it really, really faster, and created an explosion thereby of utterances. Again, for good or for ill, people in the corporate world will tell you on the one hand it's great, and on the other hand, now we're just sort of drowning in useless memos, basically.

I wasn't necessarily arguing against the fact that people should pursue efficiencies. Merely to point out that this is, if you're ever wondering what the overall pattern of how software works is, this is it.

Certainly, I think you're quite right, the correct response is probably less to say, "We shouldn't have efficiencies," ... and frankly I think it's probably best to let innovators do whatever the heck they want, you know, within reason, and have society try and organize its responses to it. Rather than to say, "Don't do this in the first place."

Julia: Oh yeah, I agree with that. I wasn't sure, reading your book, if that was your position or not. So, nice to hear it articulated.

Clive: Well, I mean, I think it's partly because I wasn't trying to write a big didactic manifesto. When it comes to the free market, I'm raised in Canada and I still have an essentially Canadian view of this, which is like -- what you want is a dynamic marketplace, that has an active state that organizes how to deal with the sort of problems [caused] by it, basically. That's the way I approach all these things.

I do think that there are still... one of the things that I think I probably do come more down negatively on the side of, is the problem of scale in large technology companies. Which is to say: through a concatenation of influences, ranging from the dictates of how venture capital funds things, and why it funds them. Ranging to the fact that software itself is just a really interesting new form of machine, in that it can scale far faster than other machines could -- if you and I put together a tractor company, there's only so fast we can make tractors. But when Instagram rolls out stories, there's very little marginal cost in the thousandth versus the millionth versus the billionth person using it, so it can scale much more rapidly.

For the whole bouquet of these reasons that I point out in the book, there's really an interesting challenge to the marketplace -- that you get these very large companies to grow very large very quickly, and establish extremely dominant positions that are hard to unseat. There are strong first mover advantages.

I think actually one of the most interesting conversations now is whether classic antitrust actions need to be taken against the scale of some of these companies, because at this point they might be actually thwarting innovation. Because mostly, at this point in time, what innovation is in Silicon Valley is trying to create a company that one of the four big ones will buy up and either kill or phagocytose, basically. Like at this point in time, there's no one who's trying to compete with any of the major four or five big companies. It’s not at all a competitive landscape.

That's very interesting. That, to me, is a gnarly, weird problem.

Julia: Talking more about unintended negative consequences of tech -- you have a thread going throughout the book, about how the engineers and designers who built the world of social media kind of naively failed to foresee some of the negative consequences of their platforms. Specifically, one of the reasons why being that they had this kind of naïve, like, "Well, it's communication, more communication is better," kind of mindset.

You quote an early Twitterer named Alex Payne making this point. He said,: "I saw a lot of really smart people who were smart in a very narrow kind of way that didn't intersect with humanities, folks who were just interested in math and statistics, programming obviously, business and finance, but who had no insight into human nature. They had nothing in their intellectual toolbox that would help them understand people."

Clive: That's right, yeah.

Julia: So I feel skeptical that the math-y engineering mindset is the problem here. To me, it seems like the problem is just that any human designing a

product, or system, that's going to be used by society as a whole, is just going to have a bad time.

Partly just because things are complicated. Like, it's really hard to predict the large society-wide effects of a thing. And also just because, you know, humans all have our own ideas about what's obviously good or fun or bad or off-putting, or something, and then we inevitably transmit some of that to our creation.

It seems to me that this is true of just everything. Like, laws, or social programs with unintended consequences. Like DARE, the drug abuse resistance education program here in the US, I don't know if you had it in Canada. It was designed to keep kids off drugs, and it actually increased the rate at which kids started taking drugs.

I don't know, to me this just seems really hard, and I'm not convinced that people with humanities degrees would do a better job. What do you think?

Clive: Interesting. I mean, you'd have to -- basically, what you'd have to do is you'd have to find counterexamples, right?

Julia: Oh yeah, do you have counterexamples?

Clive: I think some counterexamples do actually exist, though. You know, like say, let's take a look at Flickr. Flickr was essentially a social network, although people don't think of it that way anymore. In its early days, the whole thing was: you're going to post photos, and other people are going to look at them, and there's kind of going to be people talking to each other about their photos. They actually had comment threads and whatnot.

Because a bunch of the people that were involved with Flickr were even just a tick older, you know kind of in their late 20s, early 30s. They had been around in the early days of blogging, and some of them had been around back in USENET. Many knew that the tone set in the early days of how someone uses a tool has an effect, that companies can actively decide to create cultures and not create cultures.

A bunch of people took it upon themselves to do a lot of work in trying to inculcate a civil culture, a pleasant culture on Flickr, and they worked really hard at it. Like you know, someone posted something, a photo, and they were in there in the comments going, "Hey, that's awesome, check it out, you might check this other person's stuff out." There's this deep sort of complicated, un-automatable human work that went into that, and it was quite successful.

I mean, as a lot of people went on to later describe it to me, most people think of moderation as discouraging bad behavior, but partly what they were also doing is encouraging good behavior.

Julia: How big did Flickr get, very roughly?

Clive: It got pretty massive, it got pretty massive over time, yeah.

Julia: And the did the culture of Flickr manage to --

Clive: It remained pretty good, it remained pretty good.

Julia: Wow, that's surprising.

Clive: Now, we don't know how well it would've been in the long run, because Yahoo bought it, and basically broke it. They kind of destroyed it.

It is entirely possible that that culture could've died over time, certainly, but the point being that these things are possible if you think about them. Absolutely no one was thinking about it in these companies, in part because they were so damn young. I mean, they hadn't even been around for blogging and USENET.

In fact, this is one of the interesting problems people talk about, like Silicon Valley being demographically narrow, and they usually mean it's mostly guys, and that shrinks the sort of-

Julia: Or white and Asian.

Clive: Yeah yeah yeah, white and Asian young guys, and they talk about how that shrinks the space of intellectual talent. That's true, and I talk a lot about it in the book -- but one of the things that's kind of funny I actually didn't talk about in the book, and probably should've talked about it more, is age. I mean, it's like a Logan's Run in Silicon Valley. When someone is in their 30s, or certainly in their 40s, they just get squeezed out of these companies, who don't understand what they're worth.

You can ascend the ladder to be a manager or something like that, but there's a finite number of those seats. If you want to just be an engineer and just like to make stuff, they shove you off to one side. Maybe they can't squeeze 120 hours of work a week out of you anymore, or they don't want to pay you what it's worth. So they lose all this rich knowledge, like the people who've been in that rodeo twice before. They lose all that design knowledge and all that engineering knowledge.

I'm not saying you're wrong. I think -- and this goes back to the fact that actually what I really identify as a problem with a lot of these social

networks is scale, they're simply so big that I think they just become unmanageable. I think you are correct that complex systems always develop latent problems, you know -- like the interstate. That seemed like a great idea at the time. It was a great idea at the time, it caused enormous economic activity in the country. But we baked the planet, basically, with it too, by encouraging an unreal amount of driving that is really, really hard to step off the treadmill of now. You know, complex systems are complex, you're not wrong about that.

Julia: You also wrote about a different aspect of coding culture, focusing I guess mainly in the tech companies in the 90s and early 2000s. Which was this super confrontational, blunt, no bullshit, super intense culture. Where people would like throw chairs across the room when they were frustrated with their smelly code.

I'm curious, would your position be that that is something bad or something to be avoided – like, companies should be pressured to not have cultures like that, because it makes it really hard for people who are not comfortable in that culture, especially women?

Clive: I am absolutely in favor of that. I don't see any damn reason to tolerate people being dicks, absolutely zero reason. Nobody needs to behave that way.

Julia: I'm going to try to make a counterargument and see what you think of it.

Clive: Okay, go for it.

Julia: First of all, I want to distinguish between "dicks" in the sense of like, you know, actively harassing or backstabbing, or genuine jerks... I want to distinguish that from the kind of culture where it's just acceptable for people to raise their voice and say, "This is bullshit, or this code sucks," or whatever. Where it's just understood that that's the culture, and the people who are comfortable with that are not bothered by it, because they get it, they don't perceive it as an attack on them.

The counterargument that I want to put forward -- which is not mine, I saw someone else make it online -- is that to really have a maximally inclusive tech world, what we need is a diversity of different kinds of office cultures. Where the people who want a very pleasant and civil and neurotypical workplace, there's lots of places that have those.

Then the people who can't function well in workplaces like that, and are just going to get on other people's nerves because they have no filter, but they're still really valuable programmers and could contribute to society, also have workplaces that they can go to where they can thrive.

We just want to be maximizing the different kind of workplaces out there. As opposed to trying to get every workplace to be the one that hits the most people, or works for the most people.

Clive: I mean, I suppose that's a perfectly fine goal, and I guess there's nothing to argue with there, except that that's not remotely the situation we seem to have. I mean, particularly in the area of software, there is this overly-romanticized veneration for the sort of brilliant jerk. I think in some respects, it's kind of funny -- one of the things that comes up in my book a couple times, you probably noticed, is a comparison between poets and coders, right?

Julia: Yeah.

Clive: They both work with precision in language, they both prefer to have like 10 solid hours where you don't bother them. In fact, and I make this case I think in the book, you really shouldn't bother them. Like, you should let them just not be bothered while they do that work, because it's this deep, deep, mental, intense work that requires immersion. They're building a castle.

You also get this sort of self-aggrandizing, self-romanticizing bullshit about how tempestuous we are. It's crap, it's crap. I've organized my crew, I've worked in places where I've been around people like that.

Julia: I can't stand people like that, personally, I will avoid them.

Clive: I've seen no evidence, I've never seen any evidence that those places are uniquely productive compared to others.

Julia: Yeah no, uniquely productive was not part of the --

Clive: I'd like to see the data, I've never seen the data on that.

Julia: You're talking about the places that have a pretty normal culture, except they have like one or two brilliant jerks or something?

Clive: Yeah. The truth is, when you talk to the really talented people, like the Jeff Deans of the world, like the ones who have tilted the universe on its axis with the quality of their software, they're incredibly awesome people. You know, probably because they understand that at this point, software is like completely a team sport. Almost no one does anything on their own, and so if you can't actually work with other people, you're going to dramatically limit your ability to have a serious impact on the world.

Yeah, what can I say? I think there's an awful lot of people making excuses for their unwillingness to do any sort of ability to work with other people -- and unfortunately, management that buys into that romantic nonsense.

By the way, in terms of neurotypicality, this is also something that does not appear to me to decline on neurotypicality. I have been writing about software developers for 20 years, many of whom are not at all neurotypical, and personally I find a lot of them incredibly delightful to deal with. They're fantastic, they're incredibly perceptive and whatnot. One thing that someone said to me, and I think this is probably true, that there's a large chunk of people who are completely neurotypical, but are just assholes that claim they're not neurotypical. Frankly, giving people that are genuinely not neurotypical a bad name.

Julia: Yeah, that's really tough.

This relates to a confusion that I had about the book, which is: You mentioned this idea of a “10X coder,” and this sort of debate over whether there is such a thing as a 10X coder. I'm confused about what exactly the disagreement is, and why people care about this question. Can you explain the debate?

Clive: Yeah, sure. I think in some respects, it's probably one of the weaker chapters in my book, because I was struggling to figure out what to think about it myself.

Julia: Okay, so you're confused too.

Clive: Yeah, no no, yeah. I mean, I did my best to think through it, but I think the residual aspects of my confusion are evident in the writing.

Here's the thing about this story of the 10X coder, is that there was this historic idea that some people were remarkably better at coding than others. It was originally observed in a series of studies in the '60s and '70s that were not terribly statistically valid studies, but it kind of entered the idea that they had demonstrated that some coders were 10 times better... When I say "better", I mean measured by how quickly they can write functioning code, or how efficiently their code works, or how quickly they could find a bug.

This sort of entered the lore. Partly because on a practical basis, it certainly did seem like some people were more productive than others. Again, because of the, going back to this poetry metaphor -- if you need a poem written, a really good poem, and it's not getting written, you don't solve it by adding 10 more poets. It's an insight problem. There's kind of one poet that has to figure it out.

Coding is sometimes like that, it's an insight-based thing, where the best thing might be to find someone and just leave them alone to work on it. If you throw four more people at it, you're going to just add a whole bunch of communication complications that are actually going to slow down the work.

There is a situation where one person can have a really large impact on solving a problem, or creating something new. Like when you're really starting a project, when something doesn't exist and someone just has an idea for it. That's what they often call -- sometimes, not often, sometimes I've heard called a "greenfield" situation. Like there's a green field, and that thing's done. The person writing the prototype really punches above their weight, because they're creating all this stuff new themselves. They can move very quickly, partly because they don't have to deal with existing code, they don't have to work with stuff that's there, they're writing it all themselves.

When you look at the origins of a lot of these very epic-making pieces of software... Like Photoshop, which is like two people. Or you know the first 3D graphics engines for video games, really one or two people each time. Or the teams that created a lot of big pieces of software. Basic for Microsoft, written by Bill Gates and a team of three... You begin to believe, and I think in a way it's partly true, that identifying these core, super-talented people is the way to get good software written.

There's an aspect to that that's true. But as with all sort of self-mythologized aspects, it gets blown out of proportion. And a lot of people that merely, for example, write a lot of code in a prototype, get thought of as a 10X coder. But it turns out that when you actually have to make that prototype something that can scale out to millions of people, you have to tear it apart and start all over again. Now you have to have a team of people working very slowly and patiently on it.

And you might look at them and say, "Well, they're 1X coders," but they wind up producing the thing that works really well and robustly. They had to slow down, they couldn't just jam things out in that sort of frantic miasma of creativity. They wouldn't get the street cred and the plaudit for having created this fantastic prototype, but they also produced something that's far more reliable.

I think part of my confusion came from the fact that the idea of the 10X coder seems true in certain situations and seems very untrue in other ones. And even harmful, because you can wind up thinking that software gets made by the heroic activities of one or two people, and not creating an organization that actually says, "No, we need 50 engineers here, and they're all going to have to accommodate everybody carefully, and move slowly, and talk to each other about what the heck is going on. Because we

can't have one person being the big hero, because if they get hit by a bus, no one knows how to fix this stuff."

I think the confusion in my book is that I didn't do as good a job as I should've at articulating the dynamics of how team-based a lot of modern software is. I think I fell myself a little bit into the narrative trap, and joy, of looking at individual people, you know what I mean?

Julia: Okay, so to make sure I understand: It seems superficially like a debate about the distribution of programming talent, or something, and whether there are these few superstars. But it's actually a debate about the nature of programming work, and how far you can get with individual programmers as opposed to teams. And the reason that this is a more heated debate than it might seem like it should be is that it turns into a debate about, "Should we valorize these brilliant jerks, should we hero worship people?" Some people like that myth, and other people don't like that myth, and that's why they fight about it.

Does that seem right?

Clive: Yeah, I think you're probably doing a better job of that than I did in my book.

Julia: Well, I had help from you, so it's not a fair comparison.

Clive: When the softcover edition comes out, I'm going to be like, "Actually, Julie did a great job of articulating this," so let me just cut and paste what you said there. That's very well done, actually.

Julia: Good. Cool, so I wanted to ask you about your earlier book, which is called Smarter Than You Think -- it came out about five years ago, is that right?

Clive: Yes.

Julia: The thesis was basically that the internet is making us smarter, in the sense of acting like this kind of auxiliary brain, enabling collaborative learning, and all these other things.

The book had a pretty upbeat, techno-optimist attitude, in contrast to some other books that were coming out at the time. Like Nick Carr's The Shallows, which was basically, "The internet is making us stupider."

I'm curious -- well first off, I'm curious if you still agree with that book, or if your perspective has changed at all.

Clive: The answer is, yeah, I actually agree with the book quite a lot. In fact, I know because I wondered, "Do I still agree with the book?" So I read it.

Julia: Oh, good.

Clive: I read it recently. There's an interesting reason why I agree with the book, which is that when I sat to write the book, I was interested in cognition, and people's fears that using technology would make you stupider. I tried to define, in a way that made sense to me, "What are some of the aspects of what our daily cognition really looks like?" Which is: Our ability to encounter information; our ability to retrieve it; our ability to make sense of it; to connect dots; our ability to externalize our thinking, and show it to other people; our ability to collaborate, and to coordinate, and to coordinate thinking with other people. Oh, and also to think in different modalities, like to think using things other than just text, which had been the dominant mode of communication for a few thousand years.

Over and over again, the story kept on finding that yeah, this is honestly very often a net good.

Now, I think the thing that I actively didn't tackle, is that I didn't make any moral argument. I did not say, "This makes you into a better person." In fact, in my politics chapter, I very explicitly pointed out how when autocrats figure out how to use technology, they become better autocrats. They become better at suppressing dissent and become better at crushing people.

I began to realize when the book came out, after I talked about it for a few years, that everyone would talk to me about the book, would fundamentally ask me, "But doesn't this make us worse people?" Or, "Can't it also make us worse people?" My answer was always, "Yes, of course, absolutely. Many, many very smart people are absolutely terrible. Many people that don't do cognition intensely are delightful and wonderful, and I would trust my son to them, in a way that I would not trust him to many, many wickedly smart people."

Which is to say that intelligence has no innate moral dimension. One of the things that, when you look at the online world right now, many of the problems we have is that some absolutely terrible people have become incredibly good at coordinating their activities, at connecting dots, at communicating in various media that are new and fantastically persuasive. It's quite alarming actually, right?

Julia: Do you understand what exactly your disagreement is with the people who are writing the more worried/pessimistic books about the effect of tech on our lives, or our psychology, like Nick Carr? Is it an empirical disagreement? Or is it just like, there's both goods and bads, and you just like focusing on the goods, and they just like focusing on the bads?

Clive: Well, there is an extent to which I think actually ... The other thing about my book, the reason why I agree with it, is that there's an enormous amount of caveats in it which most people sort of ignored.

One of my favorite reviews was Jeet Heer, who currently writes for The New Republic, and back then was writing for The Walrus. He reviewed the book, saying that, "Clive's prose so frequently includes moments of shade and complication, that you could extract a book from inside his book that has the exact opposite argument." He's sort of right.

I mean, I actually think that the disagreement was often that people like Nick Carr were convinced that the previous modes of expression, print-based mostly, were so salutary and positive compared to the modes of thought that occurred with digital tools, that it was this net decline. In a weird way, I was actually much less positive about the ... I think the situation I had was: I would look at the history, I look at the past, while this print culture happened, and I see all sorts of huge problems. Like the fact that the tools for cognition in publication and meditation were so restricted to so few people... if you were in the 18th century, I'm sure it seemed great if you were Alexander Pope, but if you were any of the other people --

Julia: It's like looking back on the 1950s and being like, "They were so great!" Well, if you were white and male and straight...

Clive: The joke I often made is something like, Nicholas Carr was looking at the glass and going, "This glass is half-empty," and I'm going, "No no no, it's one-tenth full now. It was empty before, now it's one-tenth full." Yeah, it's really bad, 90% empty, but it was like 100% empty before, now we're 10% up. I simply had a much more dismal view of the past, you know, which made my view of the present more ...

Now the one thing, by the way, the one thing I would rewrite, that I would change, is that ... A persistent warning that goes throughout Smarter Than You Think is that these tools are great, but when they become centralized, highly centralized, and they're corporate-controlled, they almost always go off the rails. Because suddenly, the tool is fighting what you want to do with it. It's saying, "Actually, you're really just here for me to keep you engaged with clickbait so you can click on ads. Actually, we're going to algorithmically determine what you want to look at, instead of letting you pick what you want to look at."

That was, back when I wrote the book, in 2011-2012, that problem was there, but nowhere near as big as it is now. I mean, I'm a person who when, even to this day, if you were to ask me, "What are the most interesting, healthy spaces on the internet, where you actually see people doing the things that are described in Smarter Than You Think?", it's

much less often on the big, highly centralized social networks. It's in these highly more distributed ones.

Every once in awhile I just, for fun, spend an afternoon going and looking at crazy old discussion boards... Not even old, new ones, but just ones that are run on completely non-commercial software, because someone just wants to talk to the other 300 model train builders around the world. In my case, I'm a guitar player, I'm a musician, so I spend a lot of time on guitar player boards, on guitar pedal boards.

And they're just unbelievably delightful spaces, because everyone's there to talk about something they care passionately about. It's people from all over the world, Russia and Texas and Ontario, and everyone's really funny and polite, and we're diving into gnarly, complicated stuff, having great conversations. Because it's being run for, I don't know, 50 cents a month on someone's server bill somewhere. There's no ads, there's no one trying to get us to click on ads. It is a genuinely civil environment.

Maybe this is the Canadian side of me speaking here, but honestly, the free market tends to break social online activity, when it starts to try and beat it hard enough so that money bleeds out, is what I've learned. If I were to go and rewrite the book, that's what I would say a little more directly.

Julia: That's an apt metaphor.

Clive: Super upbeat way to end the interview.

Julia: Okay Clive, I have one final question for you, which is: what book, or multiple books if you want, have been the most influential on your life or on your worldview?

Clive: A couple, let's see... one really big one, the one that actually I think literally made me become a technology writer is called The Real World of Technology, by Ursula Franklin.

What it really is is a collection of talks she did. The Canadian Broadcasting Corporation has this thing called the Massey Lectures, where every year they get a public intellectual to do a bunch of lectures on a particular subject. Back around 1990 or '91, they invited her, Franklin, who is a metallurgist by trade, she's a professor of metallurgy. But she'd also been someone who'd been thinking a lot about the social implications and political implications of technology.

She did these lectures, and I missed the lectures, I didn't hear them. I just saw it when the book came out, I picked it up and I read it. To me, it was incredibly interesting. Because I had become a student journalist with the idea that I would go out and would write about things that were

"important". For me, what that meant was politics, I'd go ahead and write about politics. Because that's what serious, important people write about.

It was towards the end of my degree, and I was a news editor at the campus newspaper, and I read this book. And my head sort of lifted off. Because she would sort of talk in this really smart, intelligent way about the fascinating cultural and political implications of digital technologies.

I was kind of a neophyte. I had not read the long literature that existed in this area, that goes back decades. She was introducing it to me. And that, more than anything else as a book, made me realize, "Oh, I should take the nerdy stuff I already care about, which is computers and technology and all this weird BDS stuff, and I can devote my life to it. And it will allow me to talk about all the other things that I also care about, like culture and the arts and politics and the economy and business, and stuff like that." That book is literally responsible for why I've done that.

Julia: That's so cool. And I want to encourage listeners to consider generalizing that conclusion -- to see what other fields you might not have considered you could treat seriously, or devote a serious study or analysis to, that you might actually be able to. Anyway, go on.

Clive: Yeah, absolutely. A couple other books that were catalytic was, I would say Northrup Frye's The Great Code. Which is his book on the myth and poetic structure of the Bible, basically, and its impact on literature.

I primarily studied poetry at college, like an absolute crap-ton, and I mostly was interested in pre-20th century stuff, like 19th, 18th, 17th, 15th, you know 13th century stuff. One of the things I loved about the book was that, first off, he's such a wonderful prose stylist. Like it sort of taught me, "Wow, this is what confident, intelligent writing is like," basically.

He also has such a wonderful command of history, and the history of culture, that it also made me realize that, "Wow, whenever I write, I want to have one eye on history and culture," basically. He has this great phrase where he talks about mythology and why we have myths, and he says, "News and facts are the things that are happening, or that happened. Mythology is what happens. It is the template to the things we all go through all the time, it is the sort of platonic shape that lurks behind our individual experience that helps us make sense."

That was an amazing book to read. And again, it also sort of... you know, I had always been interested in the literature of antiquity, and it sort of gave me permission to wallow in it. Because I began to realize its deep relevance to understanding the modern condition. To this day, I still like to read ancient Greek tragedy and stuff like that, because I love it. I find it revealing, I find it fascinating to think about the similarities and

differences between people over thousands of years. I think his book was enormously catalytic in giving me permission to be interested in that the rest of my life.

Julia: That's great. I particularly like how both of your suggestions, what you got out of them was both on the object level of what the book was about, and also on the meta level of like, "How is this author approaching this subject, how are they writing about it and thinking about it?" That you got value out of it on both of those levels.

Clive: Yeah, both of them have sort of a meta-cognitive aspect to them, it helped me think about my own thinking in a more clear way. Sometimes I think that's one of the most valuable things you can do to a young kid, is to encourage them not just to think about the subject matter of the thing that impassions them, but the way they think about it, and the seriousness or the levity with which they approach it. That's crucial too.

Julia: Great. Well, Clive, thank you so much for coming on the show. We'll link to your new book, Coders: The Making of a New Tribe and the Remaking of the World. Which I encourage listeners to check out if you want more exploration of coding culture and history, and/or if you want to reward epistemic virtue.

It's been a pleasure having you on the show, thanks so much, Clive.

Clive: It's been wonderful, thanks so much to you too.

Julia: This concludes another episode of Rationally Speaking. Join us next time for more explorations on the borderlands between reason and nonsense.