From Me to Team: 40-ish slides?

Learning to code collaboratively

# Hi! I’m Victoria.

I’m a PhD student in fetal physiology at the University of Auckland. I’m an aspiring scientist, a grad teaching assistant, a self-taught student of R, and I like making stuff.

As a doctoral student, my activities are grouped into a few buckets: acquiring data in clinical/wet-lab work, analysing data, writing it up and presenting it. I picked up R out of necessity more than anything else: I’d done everything manually until the volume of data and the number of mistakes I’d make and catch and make again were such that it wasn’t tenable any more.

I think that served me adequately in a large part of my degree, because a lot of the doctoral experience is highly individual. [tiny point big circle picture] So the first time I had to share the guts of my work – scripts and so forth – I felt very vulnerable. The first time I edited group work I was petrified I’d break something. And while my particular experience might be unique to my field of science, I don’t think that feeling is particularly unique.

I’d like to speak a little bit about my journey with R as an autodidact, using it for work, and how that informed my approach to working in a team environment. What this isn’t: I’m not going to speak to managing a team of data folks here; here’s a lot of different ‘team techniques’ that have fancy names: pair programming, dojo, mob programming… my insights, such as they are, are much more localised: taking the leap from solo to teamwork.

# The background: self-taught R

# The background: playing well with others

The work I do when I’m not a doctoral student – the graduate teaching work – was really the only time I was ever looking at someone else’s work. And even then, I generally have an idea of what to expect. In every other situation, the work I produce is for my eyes only. Even when we get to the presentation stage, for the most part, people just want the very end product. The results of the analyses. [Nature publication formatting with methods in tiny print at the end.]

As I’ve spent more time on degree, it became increasingly obvious that other people were going to need to look at my work. At a broad level, in science there are some compelling reasons:

* Project S
  + Open-source science – how did you collect your data
    - Highlight this and zoom:
  + Open data (how did you do your analyses)
    - Highlight this and zoom
* Reproducibility
  + Reproducibility project
  + Others use same method/analyses with their own data
    - Highlight this and zoom
* Publication requirements

These are specific to science, because that’s my background. However, there’s a bigger theme here that became very apparent as my research objectives firmed up: these are problems that will require multiple people’s brains. To tackling these problems and discover solutions, we’re going to have to work together.

My first reaction was: this is concerning! Particularly as a self-taught R student. Without a formal education, I’ve just picked up things as I needed them. I just do stuff until it works. Sometimes the scope changes. Sometimes the whole approach needs an overhaul. Efficiency is often an afterthought.

I don’t know what I don’t know!

That’s OK. In many ways, that’s exactly why team work is dream work. (You don’t have to fix this alone!)

Being afraid to show someone your wardrobe: When I was doing analyses alone, my scripts were organised much like my garage. They would get messier and messier, stuffed with more and more things I don’t use anymore until I had no choice but to tidy it. But as long as the analyses ran and gave me the correct output, I didn’t care how it looked.

When I started graduate teaching, I realised I needed to have more empathy for people who might also need to use the garage.

In my first research assistant role, prior to my doctoral studies, the data pipeline needed streamlining: grabbing the data was the first part. Then it needed to be put somewhere, in a universal format. Then it needed to be cleaned and organised. Then it needed to be analysed, and those results had to go somewhere. Finally, this all needed to be reproducible for anyone else who came along after me. But I essentially had free reign to change things up, work within. When the project became larger, and I was working with someone else, making the pipeline efficient was waaaay less important than making the pipeline consistent. TEAM APPROACHES NEED COMPROMISE

Consistency > ‘better’ (usually)

In analyses, if you organise something in a particular way, and use a particular flow, it’s usually better to stick with flow than try to optimise. (if it ain’t broke, don’t break it/fix it) If you’re acquiring data, it’s a different ball game – efficiency and flexibility in workflow may have greater priority. Know your beast.

The CRAPL license

*Empathy emerges when we have the opportunity to see how our work impacts other people. No wonder collaborative coding is such a great way to build it.*

*Peer code review — the act of checking each other’s code for mistakes — calls on us to exercise empathy. As the reviewer, it’s important to recognize that someone has gone to considerable effort to write the code that you are about to critique. As such, try to avoid using phrases that might imply judgment or trivialize their work. When you refer to their code, you want to show them the specific functions and lines that you have questions about, and suggest how they might refactor it. Sharing learning resources can also be more helpful than spoon-feeding a solution. Some of the most useful feedback I’ve received from code reviews has come in the form of educational articles, videos, and even podcast recommendations.*

*Writing good documentation for your code also goes a long way. An act as simple as creating a readme with clear installation instructions shows empathy for anyone who needs to work with your code. GitHub founder Tom Preston-Werner advocates a readme-first approach to development.*

*“A perfect implementation of the wrong specification is worthless. By the same principle, a beautifully crafted library with no documentation is also damn near worthless. If your software solves the wrong problem or nobody can figure out how to use it, there’s something very bad going on.”*

*— Tom Preston-Werner, GitHub Founder*

*I’ve also spoken with tech founders who treat documentation as an essential part of successful onboarding. One CTO said that if a junior developer struggles to reach a level of productivity within six months of joining his team, it points towards the codebase not being well documented enough. It only takes a few seconds to add an explanatory comment to a complex function you’ve written, but it could save the next person who joins your team hours of effort.*

Naming stuff: traditions would suggest that you must have short names. I advocate for informative names, all the time. Most IDEs will autocomplete your really long variable names without much effort on your part.

Self-documenting code: R is nice, in that a lot of stuff is human-readable off the bat. Generally, I err on the side of putting in more comments rather than fewer.

**Matthew Might: the CRAPL**

[**https://matt.might.net/articles/crapl/**](https://matt.might.net/articles/crapl/)

**Lisa De Bruin** <https://osf.io/4i578/>

[Everything Hertz episode](https://share.fireside.fm/episode/fVsoDikh+d7EpT3fz) <https://share.fireside.fm/episode/fVsoDikh+d7EpT3fz>

**Edwin Thoen:** [Solving the Path Problem](Solving%20the%20Path%20Problem:) [**https://edwinth.github.io/blog/multiperson-project/**](https://edwinth.github.io/blog/multiperson-project/)

Some of the preparation I did (because science teaches you to be as prepared as possible before you get smacked in the face) [Mike Tyson quote/picture]

In the before times, a hack-a-thon type event was a good place to get together, as were the R-ladies meet-ups [going through tidy-Tuesday was really informative]. Obviously there’s more considerations around this than there were previously

Dipping your toes into collaboration can mean contributing on GitHub. Go fork something and see if you can read someone elses stuff. What’s easy for you to pick up? What’s difficult? Why?

Something that I am going to pinch from Souxsie Wiles is that you don’t have to be all things to all people in order to contribute. [Everything Hertz more janitors episode]. You can modify a wiki, or update documentation – to do that, you’ll understand the coding skills and be able to communicate how to use them.

Bonus round: empathy. If you ever do end up in a pair, make your mouse a little bigger than it would be otherwise. We are all staring at screens more, for longer, than we would usually. People are tired; make it easier for them.

*From the millions of man-hours that go into making CGI movies to the intense development crunches leading up to big-budget video game releases, towering technical achievements take a mind-boggling amount of effort. The first time I saw my current employer’s codebase, I was floored by the enormity of it all. How on earth did anybody build this?*

*The answer is that everybody can build a lot more than anybody, given the right collaborative framework. In companies that encourage collaborative coding, the software doesn’t emerge from the efforts of a lone genius. Instead, there are ways of working together that help great teams to do amazing work. Developers at Founders and Coders practice a popular software development methodology known as ‘Agile’, and in my experience, it puts the ‘functional’ in cross-functional development teams.*

*Entire books have been written about Agile, but here is a summary of the core concepts:*

*A product development team breaks down large pieces of work into small units called ‘user stories’, prioritizes them, and delivers them in two-week cycles called ‘sprints’.*

*For as long as the project continues, the cycles repeat, and new product requirements get fed into a backlog of tasks for future sprints.*

*The team holds daily standup meetings to discuss their progress and address any blockers.*

*The process is both incremental and iterative: the software is built and delivered in pieces and refined in successive sprints.*

Everything Hertz Lisa DeBruine (episode 78) LISA DE BRIAN

Patchy stuff on OSF

Dirty code is better than no code. Cleaner the better, more comments is better

No-one generates fully formed clean stuff on the first go. Going in with that expectation means you’ll just never do it.

There’s also an embarrassing idea of ‘oh, and now I have re-discovered the wave function’. Well, that’s great – now you know it inside out. Just because someone else happens to know a function that does the same thing, and it’s in a package somewhere, doesn’t mean that what you’ve done is a waste of time.

As well, a team will often be able to search for what you need (that package, that function) using different keywords or whatever – pooling knowledge can lead to some efficiencies.