The soft side of code review*

*and why it matters

Sadie Bartholomew
Computational Scientist
National Centre for Atmospheric Science (Computational Modelling Services group) & Department of Meteorology, University of Reading

RSE South Workshop
Theme: Sustainability and Reproducibility of Research Software
16th May 2023, University of Reading

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

University of Reading

Background: about me

- I work here at Reading for the Dept. of Meteorology and a UK distributed research centre, NCAS
  - Meteorology ≈ atmospheric / weather & climate science
  - NCAS is one of six research centres* supported by NERC (the Natural Environment Research Council)

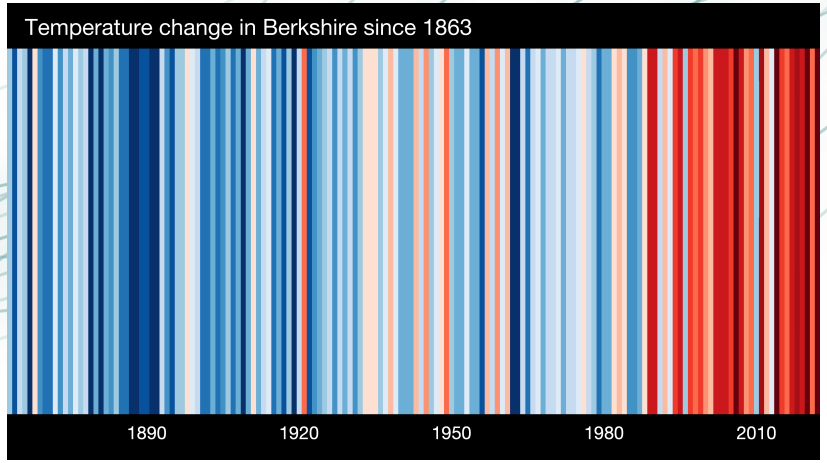
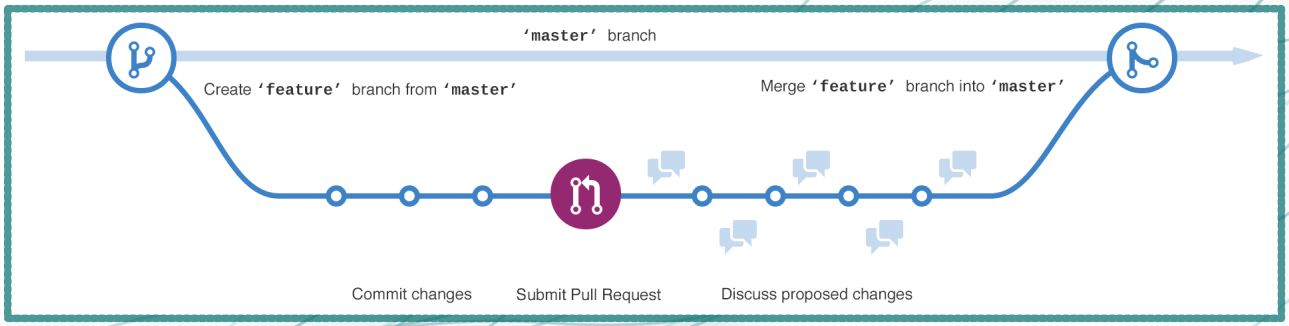


Image to left: me at Météo-France in Toulouse
Image to right: 'Warming Stripes' by Ed Hawkins, also NCAS and UoR (#ShowYourStripes)

*www.ukri.org/about-us/nerc/who-we-are/nerc-research-centres/

So… why code review?

- In my work, I mostly develop and maintain open-source libraries to facilitate climate research, in small teams
  - I mainly work on Python code managed via git, hosted on GitHub
  - We follow fairly standard git(-GitHub) workflows, akin to:



- As an effective RSE for >5 years(!) I have done a lot of code review, and come to appreciate the significance…
  - not just for code quality but team cohesion, knowledge exchange

Schematic: sourced from *https://gaboesquivel.com/blog/2018/recommendations-to-enhance-your-github-flow/*

Code review in the abstract

- Code review "is a process where someone other than the author(s) of a piece of code examines that code."*
  - other definitions include concepts of 'systematic examination', 'quality assurance', (a form of) 'peer review', etc.
  - notably can be formal *or informal*
- The *soft* side: **not what you say, but how you say it!**
  - reviewee can't read your mind: need to convey your findings
  - some ways to convey are better than others!

Image from *openclipart.org/detail/320248/yelling-man*
**google.github.io/eng-practices/review/*

Focus in on the *soft* side

- Extreme examples to illustrate the point...
  - Obviously, one is better than the other. How can we share specific code feedback in a way that is as clear and helpful as possible?

“ This code is ****! ”

“ This method name could be misinterpreted because of *<reason>*. How about we use *<name>* instead? ”

Ultimate aims:

none (laziness/lack of care), one-upmanship, imposing opinions

improving code(base), sharing knowledge, camaraderie

Considerations: an acrostic

- Having contemplated my 5+ years of RSE work & code review experience, I identified considerations for the soft side of code review I thought most important & formed an acrostic mnemonic. **Be a "REVIEWER":**
  - **R**espectful – notably "2 P's": **p**rofessional and **p**olite
  - **E**xplicit
  - **V**aluable – notably "2 C's": **c**ontext-guided and **c**onstructive
  - **I**terative
  - **E**ndorsed
  - **W**illing – notably "2 U's": **u**nited and **u**nfussy
  - **E**xploratory
  - **R**eproducible

REVIEWER: R (#1)

- **Respectful** – notably "2 P's": **p**rofessional and **p**olite
  - A basic minimum! Hopefully should go without saying…
  - Bring the right attitude & consider tone
  - Don't dismiss any comments, regardless of relative levels of expertise of reviewer and reviewee. Code review is a really important learning mechanism for juniors; don't demoralise!

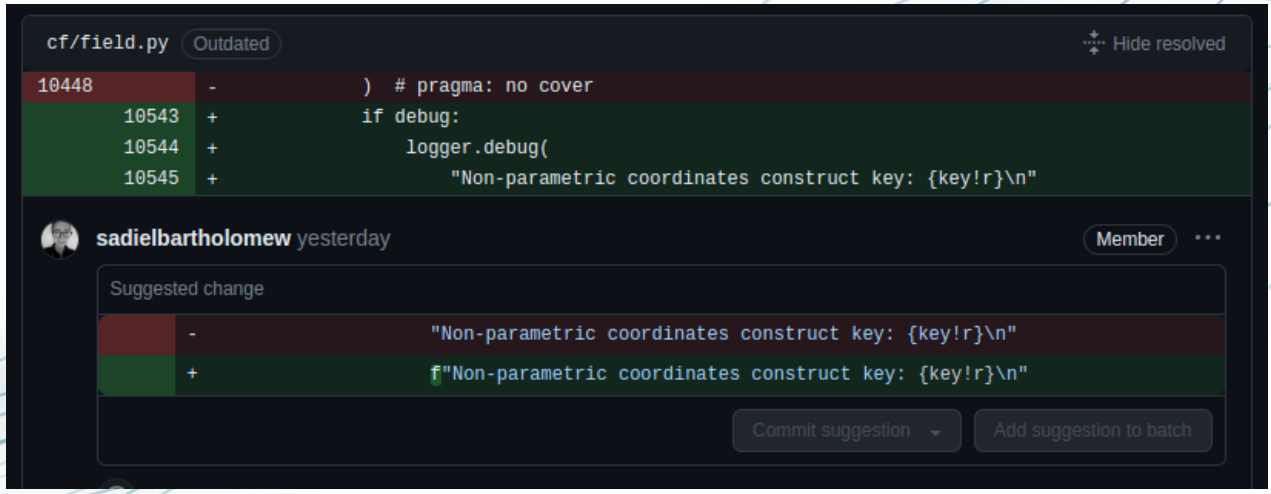
" Wrong! "

" Were you drunk when you wrote this, buddy? "

" Thanks for clarifying. With regards to… "

REVIEWER: E (#1)

- **Explicit**: communicate effectively (clearly & directly)
  - *Explain* > *Describe* & remember, the reviewee can't read your mind
  - Be specific – ideally make many self-contained comments, in-line if localised, rather than fewer and larger feedback blocks



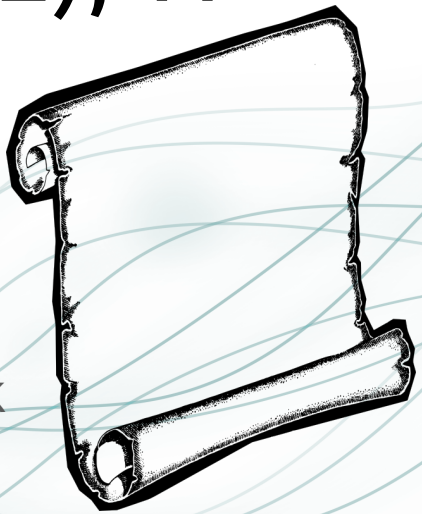
- Provide enough detail but not too much – be sufficient and concise

REVIEWER: V, I

- **V**aluable: always keep in mind the context
  - notably "2 C's": **c**ontext-guided and **c**onstructive:
  - A good idea if possible is to pre-agree the *scope* of a given review, for example as a checklist. Should it be a quick sanity check or a thorough review? What's relevant (performance, style, etc.)?
  - Consider the expertise level of the reviewee, e.g. if a senior reviewing someone more junior, avoid jargon terms and so on
- **I**terative: often goes in iterations so keep that in mind and try to reduce them
  - Back and forth with questions, a conversation!
  - Consider medium: for longer dialogue, taking it from online to in-person / video call is often very wise to reduce need for cycles

Image from: *openclipart.org/detail/171954/cycle-color*

REVIEWER: E (#2), W

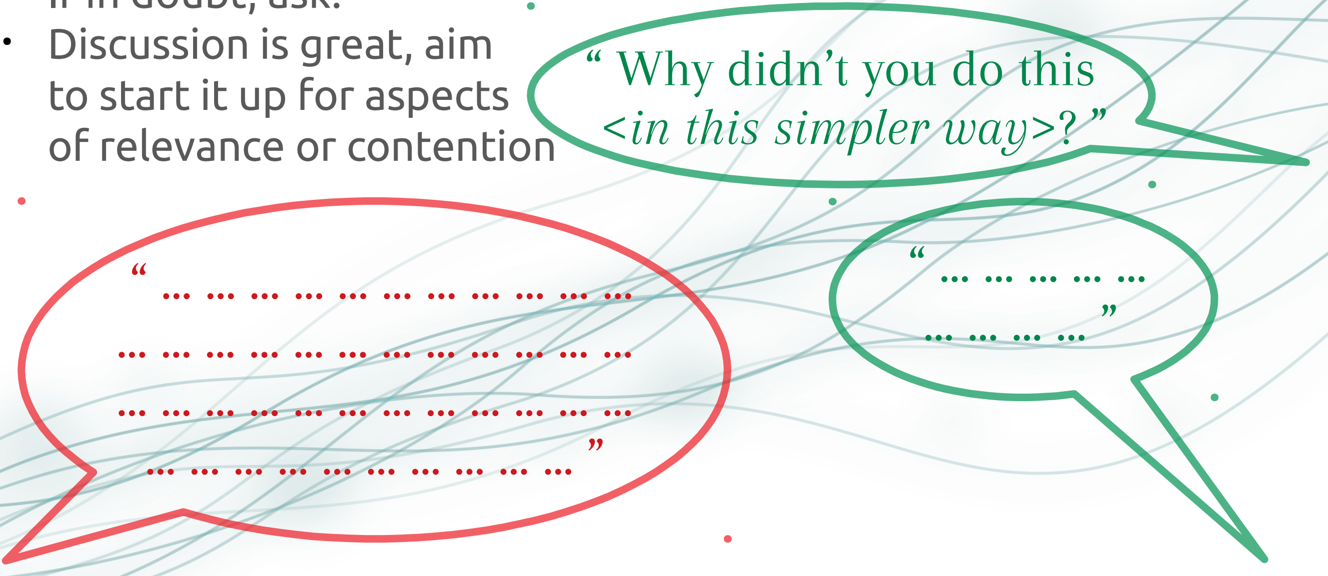


- **E**ndorsed: reference standards whenever you can
  - If you can reference a standard e.g. for correct code style that the code changes aren't obeying, that makes the suggestion uncontroversial
  - Stack Overflow threads are a staple and OK to link to!
- **W**illing: the joint goal is to improve the code so be flexible not stubborn (it's not a competition)
  - You may have strong opinions on matters relating to code, but put them aside for the greater good if there's not a consensus!

Image from: *openclipart.org/detail/327468/vertical-scroll-frame*

REVIEWER: E (#3)

- **Exploratory**: investigate & ask questions, don’t assume
  - Things may have been done for reasons you haven’t anticipated
  - If in doubt, ask!
  - Discussion is great, aim to start it up for aspects of relevance or contention



“ Why didn’t you do this *<in this simpler way>?* ”

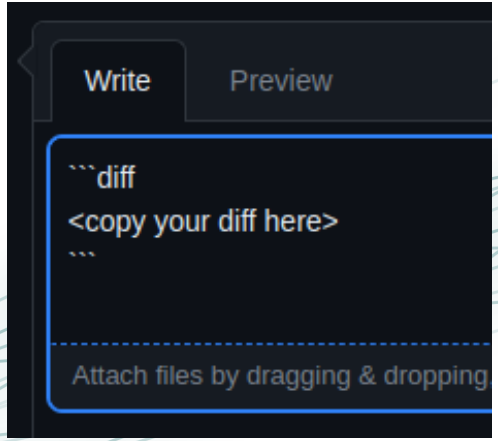
“ … … … … … … … … … … …
… … … … … … … … … … … …
… … … … … … … … … … … …
… … … … … … … … … … ”

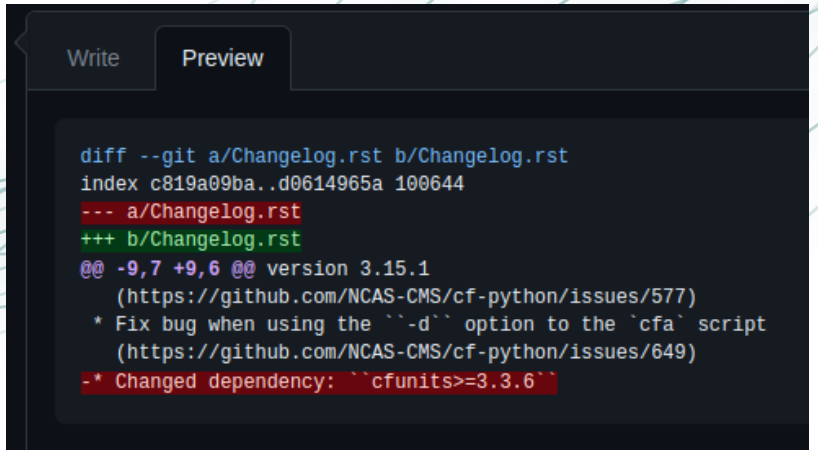
“ … … … … …
… … … … ”

*For further information, see e.g: *en.wikipedia.org/wiki/Minimal_reproducible_example*

REVIEWER: R (#2)

- **R**eproducible: use code snippets, diffs, MREs, etc.
  - As supported by UIs such as GitHub, often the best way to explain yourself is by including snippets of code e.g. suggestions for alternatives, demonstrative diffs (example below), minimal reproducible examples (MREs)* for issues with the code, etc.





*For further information, see e.g: *en.wikipedia.org/wiki/Minimal_reproducible_example*

Summary

- Code review "is a process where someone other than the author(s) of a piece of code examines that code."
  - Definitions often mention 'peer review', 'systematic examination', 'quality assurance', etc. Includes informal review and self-review!
- Highlighting the importance of the *soft* side (**not what you say, but how you say it!)**
  - Reviewer-reviewee communication in the abstract
- Introduced an **acrostic mnemonic** as a guide to be a good **"REVIEWER"**:
  - **R**espectful, **E**xplicit, **V**aluable, **I**terative, **E**ndorsed, **W**illing, **E**xploratory, **R**eproducible

Postscript...

- I am hoping take this guidance and the “REVIEWER” acrostic forward to form a blog post Software Sustainability Institute and **would appreciate any feedback you have** to help with that.
  - i.e. something for https://www.software.ac.uk/blog
  - Please get in touch with me today at this workshop, or via sadie.bartholomew@ncas.ac.uk to share your thoughts. Thanks!

SOFTWARE SUSTAINABILITY INSTITUTE