Best Practices for Contributing Code for PPI

v1.0

By Sam Kanner[[1]](#footnote-1)

Released: Oct 15 2017

# Purpose

This ‘best practices’ guide should be read by engineers who write code at PPI. It borrows heavily and directly from a paper on entitled, *Best Practices for Scientific Computing* in **PLOS Biology** [[2]](#footnote-2).

# Best Practices

1. Write programs for people, not computers.
   1. A program should not require its readers to hold more than a handful of facts in memory at once.
   2. Make names consistent, distinctive, and meaningful.
   3. Make code style and formatting consistent.
   4. Provide comments frequently, especially at the beginning of functions, sub-functions or distinct sections of the code.
   5. Only include outputs to the command window unless absolutely necessary.
2. Let the computer do the work.
   1. Make the computer repeat tasks.
   2. Save recent commands in a file for re-use.
3. Make incremental changes.
   1. Work in small steps with frequent feedback and course correction.
   2. Put everything that has been created manually in version control.
4. Don't repeat yourself (or others).
   1. Every piece of data must have a single authoritative representation in the system.
   2. Modularize code rather than copying and pasting.
   3. Re-use code instead of rewriting it.
5. Plan for mistakes.
   1. Add assertions to programs to check their operation.
   2. Turn bugs into test cases.
6. Optimize software only after it works correctly.
   1. Use a profiler to identify bottlenecks.
   2. At first, try to develop software to address a very specific problem. Then, try to think of different uses, applications, cases around your problem and try to generalize the software to take into account these situations.
7. Document design and purpose, not mechanics.
   1. Document interfaces and reasons, not implementations.
   2. Refactor code in preference to explaining how it works.
   3. Add/edit documentation for a piece of software when a major change has occurred.
8. Collaborate.
   1. Use pre-merge code reviews. For example, you can push your changes to a separate branch and notify team members of the existence of this branch. After review, the team can merge the branch with the master branch and delete the temporary branch you created.
   2. Use an issue tracking tool.

1. As well as many researchers: Greg Wilson , D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White, Paul Wilson. [↑](#footnote-ref-1)
2. Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. (2014) Best Practices for Scientific Computing. PLoS Biol12(1): e1001745. https://doi.org/10.1371/journal.pbio.1001745 [↑](#footnote-ref-2)