

# Improving our code

Mike Jackson  
Software Architect

EPSRC

NERC SCIENCE OF THE ENVIRONMENT



CRAY  
THE SUPERCOMPUTER COMPANY

epcc



# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US)

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.



epcc



## What is the most efficient way to detect bugs?

- “Rigorous inspections can remove up to 90 percent of errors from a software product before the first test case is run”
- First review and first hour matter most
- Spread knowledge around a team and increase “bus factor”
- Pair programming

- Glass, R. Facts and Fallacies of Software engineering, Addison Wesley, 1992. ISBN-10: 0321117425. ISBN-13: 978-0321117427. p104.
- Fagan, M.E. “Design and Code inspections to reduce errors in program development”, IBM Systems Journal 15(3), pp182–211, NN 1975. doi:10.1147/sj.153.0182.
- Cohen, J. “Best Kept Secrets of Peer Code Review”, Smart Bear Inc, 2006. ISBN-10: 1599160676. ISBN-13: 978-1599160672.
- Cohen J. Modern code review. In Oram, A, Wilson, G. (editors) “Making software: what really works, and why we believe it”, O’Reilly, 2010. ISBN-10: 0596808321. ISBN-13: 978-0596808327. pp. 329–336.
- Wilson, G., Aruliah, D.A., Brown, C.T., Chue Hong, N.P. and Davis, M. “Best Practices for Scientific Computing”, PLoS Biol 12(1): e1001745, January 2014. doi:10.1371/journal.pbio.1001745.



## Not just for bugs...

- “Write Programs for People, Not Computers”
- Modular structure and design
  - Break programs into files, modules and packages, classes, methods and functions
  - Every component has a well-defined purpose
  - Highly-cohesive – performs a single, easily understood task
  - Loosely-coupled – minimal dependencies on other components
  - DRY – don’t repeat yourself
  - YAGNI – you ain’t gonna need it



## Not just for bugs...

- Consistent, distinctive, meaningful, self-documenting names
- Readable code style and formatting
  - Sensible whitespace and indentation
  - Avoid cryptic code
  - Conforms to language and/or project standards
- Accurate and appropriate comments
  - Files, modules, packages, classes, methods, functions, input/output/return types, exceptions
  - Why the code is as it is



## Travelling Salesman Code

```
$ python tsp.py 7 scottish_cities.pickled tmp.dat
scottish_city_loc.csv
```

- Review code (30 minutes)
- Report back, group together issues by category (30 minutes)
- Prioritise issues (30 minutes)
- Coding standards
  - van Rossum, G., Warsaw, B., Coghlan, N. PEP 0008 - Style Guide for Python Code, 5 July 2001, <https://www.python.org/dev/peps/pep-0008/>
- Automated tools
  - `pep8` Python style guide checker, <https://pypi.python.org/pypi/pep8>
  - `pylint` code analysis for Python, <http://www.pylint.org/>

