



# Software Carpentry for RMG developers

RMG Study Group  
March 21, 2014

# SoftwareCarpentry.org - an introduction

- ❑ Teach software skills to researchers
- ❑ Provide guidance on best practices and other problems
- ❑ Our contact: Trevor King
- ❑ Familiar with git, Python, Django
- ❑ We were interested in promoting code review

# With excerpts from:



Ensuring Software Success™

## 11 Best Practices for Peer Code Review

A SmartBear White Paper

Using peer code review best practices optimizes your code reviews, improves your code and makes the most of your developers' time. The recommended best practices contained within for efficient, lightweight peer code review have been proven to be effective via extensive field experience.

Contents

- Introduction.....2
- 1. Review fewer than 200-400 lines of code at a time.....2
- 2. Aim for your inspection rate of less than 300-500 LOC/hour.....2
- 3. Take enough time for a proper, slow review, but not more than 60-90 minutes.....3
- 4. Authors should annotate source code before the review begins.....3
- 5. Establish quantifiable goals for code review and capture metrics so you can improve your processes.....4
- 6. Checklists substantially improve results for both authors and reviewers.....5
- 7. Verify that defects are actually fixed!.....5
- 8. Managers must foster a good code review culture in which finding defects is viewed positively.....6
- 9. Beware the "Big Brother" effect.....7
- 10. The Ego Effect: Do at least some code review, even if you don't have time to review it all.....8
- 11. Lightweight-style code reviews are efficient, practical, and effective at finding bugs.....8
- Summary.....9

[www.smartbear.com/codecollaborator](http://www.smartbear.com/codecollaborator)

# Discussions happen offline but aren't catalogued

- ❑ Offline discussions are efficient but people forget what was discussed (or were left out)
- ❑ Discussions online provide a history
  - ❑ 2 years later we can revisit a problem
  - ❑ others can join in with suggestions
  - ❑ we can attach code

# Code review: shorter is better

- ❑ You don't detect defects if reviewing too much at once.

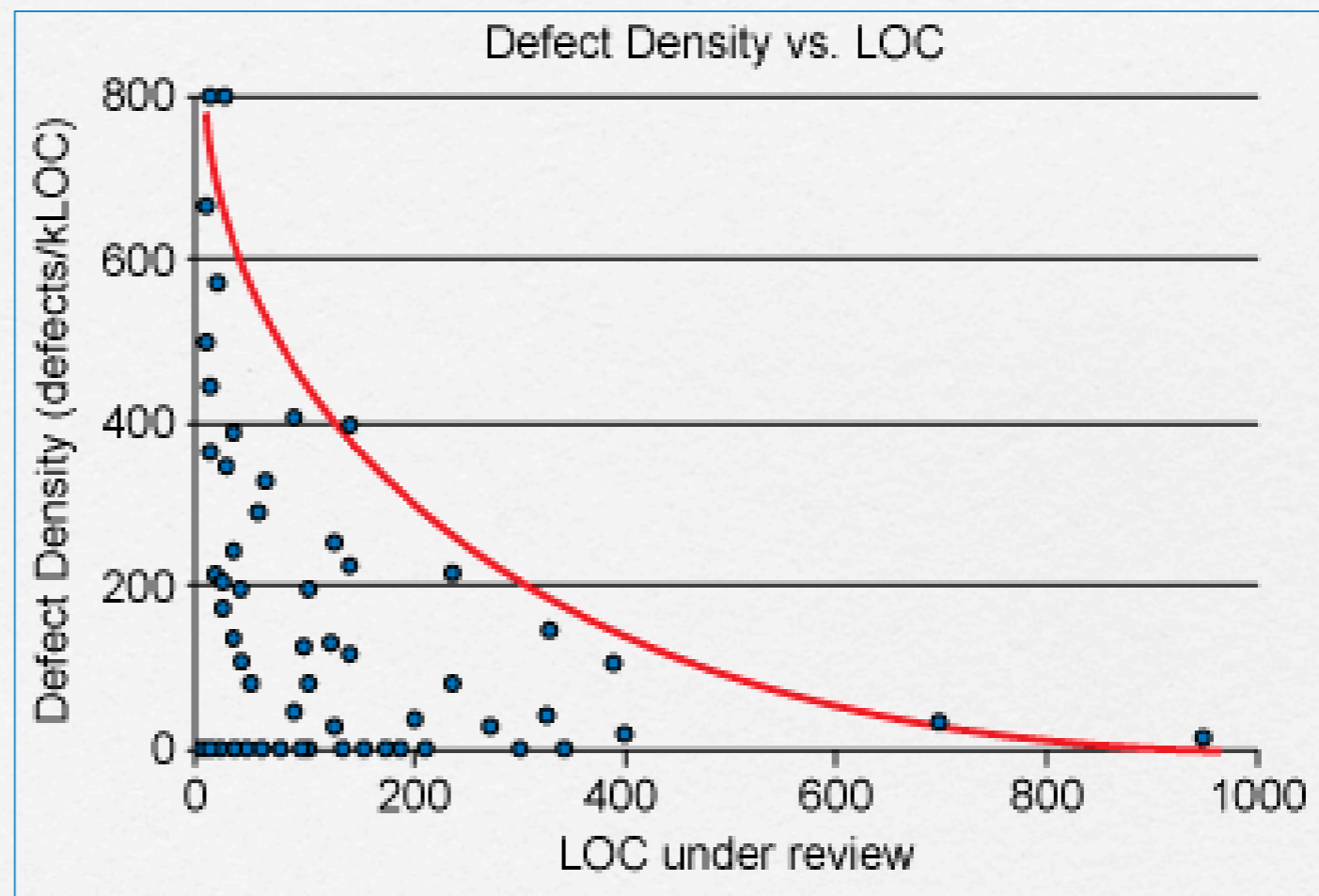


Figure 1: Defect density dramatically decreases when the number of lines of inspection goes above 200, and is almost zero after 400.

# Large commits are hard to follow so don't get reviewed properly

- ❑ Make each commit do one thing
  - ❑ GUI to help stage individual lines
  - ❑ interactive rebase if you need
- ❑ Long topic branches are hard to review
  - ❑ rebase them into mergeable chunks,  
and merge frequently (via pull requests)
- ❑ Writing good commit messages will help

# Annotation helps prevent errors

- Explaining yourself reduces your error rate!

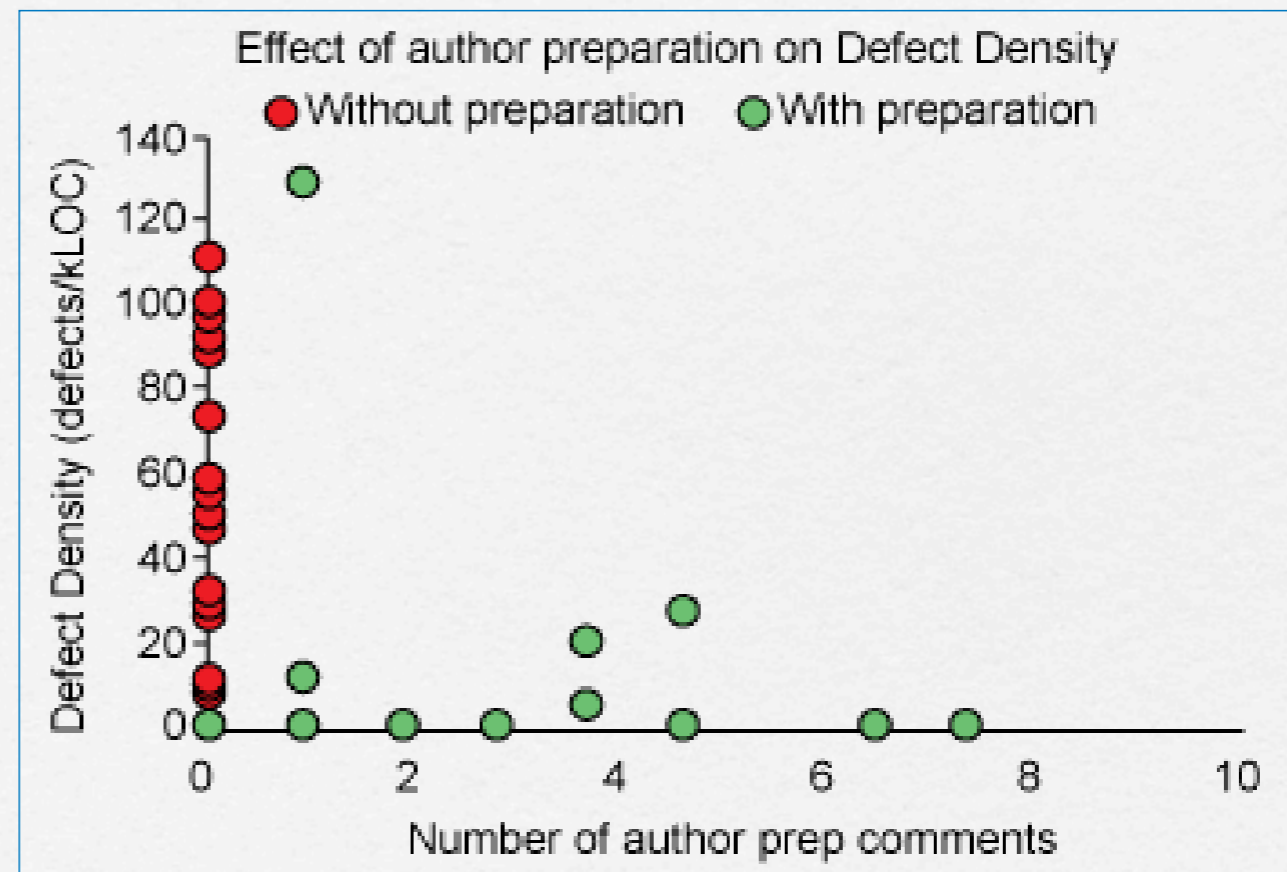


Figure 3: The striking effect of author preparation on defect density.

# Commit messages should describe *why* you're doing something

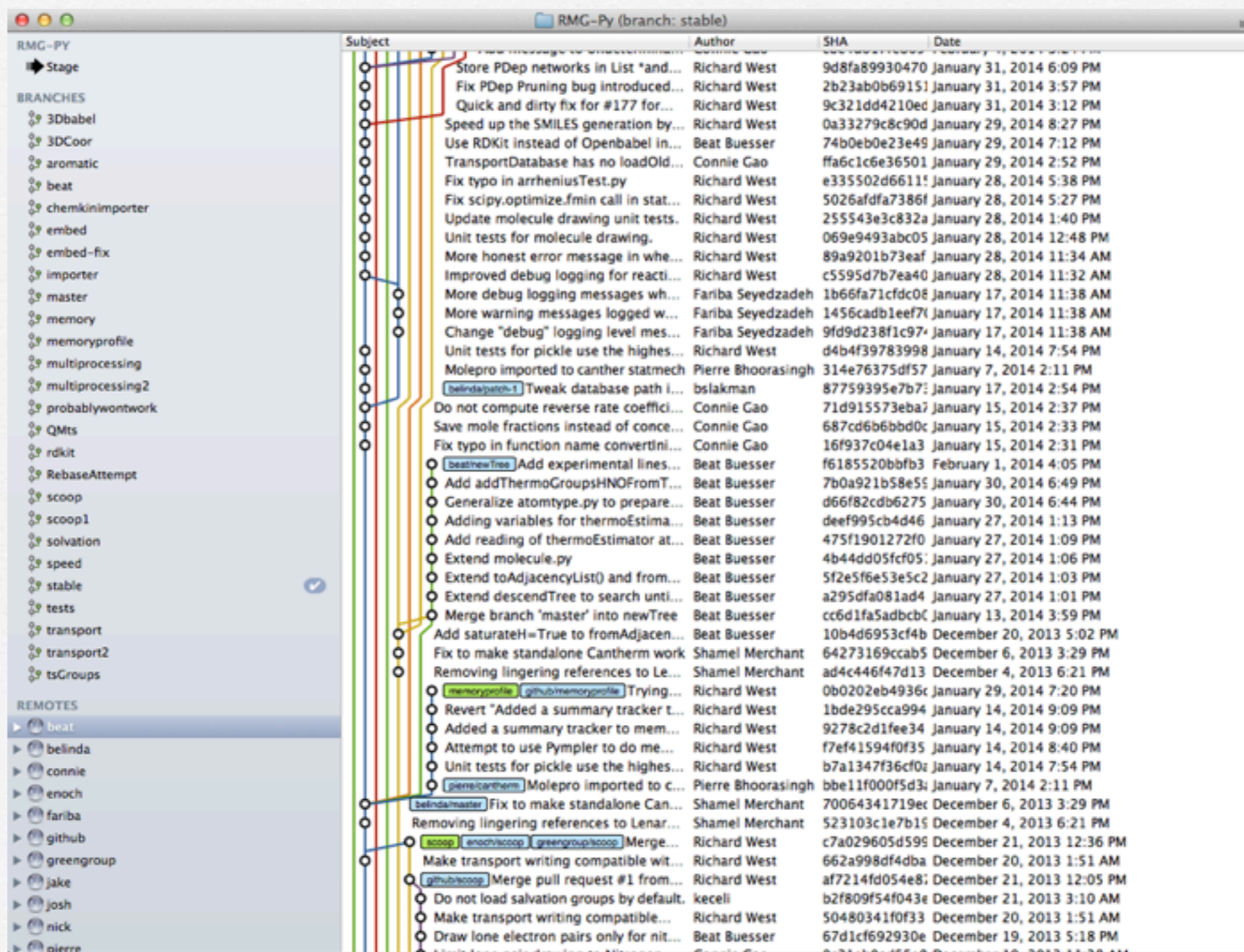
- ❑ Not just describing what was done, but the rationale
- ❑ The 'diff' shows what you did.
- ❑ Makes more sense when revisited in the future
- ❑ Helps reviewers check you're achieving what you intended, and suggest other ways.
  
- ❑ Bad: 'Attempts to fix double-ended approach'



# Nothing's too small for a pull request (or too urgent).

- Topic branches, issues, bug fixes, typos..
- Promotes discussion and code review
- Preserves discussion
- Assign more people to review and merge
  - let's help Connie out

# Festering topic branches not reviewed:



# Solution: pull requests and rebase

- Git magic to the rescue!

```
[remote "greengroup"]  
url = git://github.com/GreenGroup/RMG-Py.git  
fetch = +refs/heads/*:refs/remotes/greengroup/*  
fetch = +refs/pull*/head:refs/remotes/greengroup/pr/*
```

```
git rerere --help
```

# Problem: Topic branches remain unmerged for a long time

- Rebase often onto the master branch
  - helps when you finally have to merge
- When making core changes introduce them into the master
  - e.g. changes to molecule, reaction, etc. should be brought in early
  - allows people to discuss and adapt

# Problem: When a merge occurs we don't know if something broke

- ❑ Solution: unit tests!
- ❑ Continuous integration testing on topic branches (Travis-CI.org?)
- ❑ a study group for unittest writing?

# Better use of git and github features can help RMG developers

- ❑ Open more issues (do labels help?)
- ❑ Short commits with descriptive commit msgs
- ❑ More pull requests
- ❑ Rebase topic branches onto master often
- ❑ Merge core changes from topics early
- ❑ Write more unittests

# 11 Best Practices for Peer Code Review

1. Review fewer than 200-400 lines of code at a time
2. Aim for your inspection rate of less than 300-500 LOC/hour
3. Take enough time for a proper, slow review, but not more than 60-90 minutes
4. Authors should annotate source code before the review begins
5. Establish quantifiable goals for code review and capture metrics so you can improve your processes
6. Checklists substantially improve results for both authors and reviewers
7. Verify that defects are actually fixed!
8. Managers must foster a good code review culture in which finding defects is viewed positively
9. Beware the "Big Brother" effect
10. The Ego Effect: Do at least some code review, even if you don't have time to review it all
11. Lightweight-style code reviews are efficient, practical, and effective at finding bugs