To the Editor,

We would like to wholeheartedly thank the reviewers for their thoughtful, detailed and very helpful comments. Furthermore, given the reviewers were openly named, we have mentioned our gratitude to them in the Acknowledgements section of the revised manuscript – please advise if the reviewers do not wish to be so named.

We have addressed each point in the revised manuscript in the manner described below.

With best wishes,

Janna Hastings, Kenneth Haug and Christoph Steinbeck.

---- ---- ---- ---- ---- ---- ---- ----- ---- ---- ---- ----- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

Reviewer 1 (Robert P Davey)

1. It might be sensible to outline very briefly what software engineering is in the abstract. If the authors are targetting novice researchers, the context of the article should be expanded.

*We have changed the abstract to include an explanation of what software engineering is.*

2. "Well-operating" -> "fit-for-purpose", or "well-written"

*Fixed.*

3. The "But...." sentence should be joined to the previous with a comma, avoiding starting a sentence with a conjunction.

*Fixed.*

4. I'm not sure the focus of large datasets warrants software engineering per se. I would suggest that the complexity, diversity and relatively high availability of datasets in a variety of formats is also a very strong driver to deliver well-designed and efficient software, not to mention long-term sustainability of a tool.

*We have reworded the abstract and the background to reflect this point.*

5. For each recommendation, it would be beneficial to readers to suggest an online resource to learn more about the area.

*Given the constraint on the word count for the manuscript, rather than expanding the text of each recommendation, we have included a summary table in the conclusion which gives online resources for further reading.*

(Rec 1)

6. The authors mention checking and double-checking their code, which is effectively testing, or very closely linked to it. Consider moving the Test, Test, Test recommendation closer to Keep It Simple to keep the flow going.

*We have rearranged the recommendations accordingly.*

7. Again, starting a sentence with "So..." is a little clumsy. The previous sentence can be joined to this one with a comma.

*We have reworded the sentence.*

8. Maybe a definition of "clean" would be helpful, again to help novices understand the recommendation - it doesn't necessarily mean "simple", but "sufficient".

*We have added a footnote to a great collection of definitions of clean in the context of code.*

(Rec 2)

9. "Generic method" has distinct connotations in certain languages, e.g. Java. Using the term "utility methods" would remove this ambiguity.

*We have removed the word ‘generic’.*

10. The authors should suggest using code coverage tools that can help find duplicated code, amongst other potential optimisations.

*We have added the suggestion to use automated means to detect code duplication problems.*

(Rec 3)

11. "Coding to interfaces" is a common question in engineering, and it is very useful to mention it here. A link to some examples or extra documentation about the paradigm would be beneficial for readers (e.g. the Effective Java book by Josh Bloch has a great section on it).

*We have mentioned coding to interfaces and added a reference to the Effective Java book.*

12. APIs and their relevance here is not mentioned by the authors and the article would benefit from this.

*We have mentioned APIs in the revised text.*

13. An additional suggestion to existing library reuse would be to reinforce that using a library with more functionality than you need is not bad practice and definitely not a reason to reinvent the wheel. I often come across the argument that developers choose not to use a library because it contains "everything but the kitchen sink" and write their own solutions, which is not optimal in most cases.

*We have now included specific mention of re-use of libraries with more functionality than needed.*

(Rec 4)

14. "test software than develop it" -> "test software rather than develop it"

*We have reworded this sentence for clarity.*

15. Consider linking the modular design section to your testing section. Unit tests are very effective when coupled with the design of the way modules fit together.

*We have added a brief mention of testing to the modular design section, which now appears after the testing section due to the rearrangement. All the different sections are somewhat interrelated, of course.*

16. Consider mentioning Test-Driven Development as a more extreme development activity here too.

*We have mentioned test driven development in the testing section.*

(Rec 5)

17. Users asking for too much leads to feature creep, and bad prioritisation. It would be good to expand on what the authors mean here, or at least mentioning "gold plating" in this section to lead on nicely to the next.

*We have added specific mention of feature creep to this sentence. We hope that the flow to the next section (gold plating) is clearer now.*

(Rec 7)

18. "But there is no need..." - again, link this sentence with another.

*We have replaced ‘But’ with ‘However’ in order to improve the flow here. We have tried to avoid overly long sentences throughout the current draft.*

19. The [1] reference is confused with the use of it as a footnote. Use the dagger or asterisk here instead.

*We have switched to symbolic footnotes. However, the identical numeric footnotes and references appear to be the default in the BMC latex template so we are not sure they will accept the change.*

(Rec 8)

20. The sentence "Unfortunately, spaghetti code is alive and well" is redundant and should be removed.

*We have removed this sentence.*

21. Effective and granular logging could be mentioned here as a way to also trace what's happening in code modules.

*We have added mention of logging to this section.*

(Rec 10)

22. "But avoid the urge", and "And 'publish or perish'" - again, refactor these to not start with a conjunction. *We have refactored these sentences.*

23. The commentary conclusions are a little disjointed with the rest of the article and could benefit from being contextualised a little better. Similarly, perhaps expanding on some of the points would hit home the intentions. For example, suggesting areas where training can be received (MOOCS, Software Carpentry, paid training are all options) would be beneficial. *We have expanded the conclusion and included a table for further reading and learning.*

---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----

Reviewer 2 (Chris Mungall)

some of the points can be debated (e.g. it can be argued that some thought needs to be paid to optimization at the outset) but the point is that you should understand the rule before you break it.

*We added a proviso to this effect to the conclusion.*

1. The commentary would benefit from a "what next" section. The reader is convinced that they need to write well-engineered structured programs. They have a rough sense of traps to avoid. How do they learn more? I do not have an extensive list of resources to recommend. But as a start, I think the Mozilla labs software carpentry project deserves a mention <http://software-carpentry.org/index.html>.

*We have added a recommended further reading table to the conclusion, including a reference to the software carpentry website.*

1. citation for GOTOs: "Dijkstra, Edsger W. "Letters to the editor: go to statement considered harmful." Communications of the ACM 11.3 (1968): 147-148."

*We have added the reference.*

2. the final paragraph of point #10 alludes to 'second system syndrome' as described in the still-relevant classic "the mythical man month", which would be a useful source to cite

*We have added the reference.*

3. The recommendation to use a VCS is a bit buried in point #10, and there is no mention of VCS hosting solutions like github or bitbucket. In this day and age I would argue there is no excuse for not putting your software here, no matter how small. "Commit early, commit often" is another lesson I find cannot be drummed in enough.

*We have reworked the section to un-bury the recommendation to use a VCS and add specific mention of “commit early, commit often”. We agree that this is central to good practice in software engineering, but still believe it fits best within point 10 – we view it as a grand finale rather than being buried :-).*

4. For point #4, I like to emphasize test-\*driven\* development, which is a major point of many modern software development methodologies

*We now introduce TDD in that section.*

5. Point #8 seems to be conflating spaghetti code with poor module construction. But this is a nitpicking point, they are broadly analogous

*We have reworded this sentence to clarify the analogical nature of our discussion about module interrelationships and flow.*