A Checklist for Engineering Changes

These are 9 steps which colleagues should be able to observe for each piece of engineering work. In every step it is essential to pause and reflect for a moment before moving on: *Is there anything I could improve upon thus far?*

☐ **1 Demonstrate understanding** from all the necessary perspectives.

- Initially verbal and/or IM, then emails, diagrams, and code snippets. After a discussion, send detailed notes to everybody involved.
- Everybody involved should feel confident that their views are being considered, and everybody should see everybody else's point of view.

☐ **2 Explain the pros/cons** of different approaches.

- In your own time, write text and draw diagrams in your logbook.
- Later, especially in reviews, you'll be asked for evidence that you've chosen the "best" option.
- Sometimes, where people hold strong opinions, they will demand detailed answers about why their favourite option wasn't chosen, so recording this will be essential to your argument.
- Relatively informal material is easier to sanity-check than formal material like source code.

☐ **3 Describe the changes** you will make in your preferred approach.

- Again in your own time, draw diagrams and write text, pseudocode, etc. in your logbook. All forms should highlight before vs after.
- This should be more detailed and formal than the comparison of approaches.
- Later, in making the changes (step 5), the source code should be a straightforward implementation of these diagrams and/or pseudocode.
- After making the changes (step 5), you can refer to these notes when writing commit messages.

☐ **4 Explain the checks** you intend to perform on your changes.

- In your own time, write text in your logbook. This will usually include shell commands.
- In the process of thinking about how to check that your changes actually do what you indend, you may realise you need a different approach.

☐ **5 Make the changes**, i.e. modify source code.

☐ **6 Demonstrate your checks** have been performed.

- Write in commit message bodies and pull request descriptions. The body of a commit message is separate from the title. Similarly, the description in a pull request is separate from its title.
- Your colleagues don't want to read reams of shell output, so you should provide a succinct analysis for them.

- Textual output from a tool may be copied in textual form, i.e. a screenshot is not appropriate as it is not easy for your colleagues to search.

☐ **7** Explain how others can **repeat your checks**.

- Write in commit message bodies and pull request descriptions.
- Precise instructions often include shell commands and version numbers.
- You can make things easier for the reader by marking commands with backticks (rendered as teletype on BitBucket, GitHub, and other tools).
- Your colleagues should have a clear impression of the situation before your change, how your changes affect it, and what to expect after your change. Important details, e.g. error messages, must be recorded carefully.
- If there is still undesired output from tools, then include that too so that they aren't surprised, and possibly annoyed, when they rerun your checks.

☐ **8** **Describe the limitations** of your changes.

- Write in code comments, accompanying documentation, commit messages and pull requests.
- If everything went to plan, then you can refer to your notes from step 2 and step 3. Otherwise, it is even more important to communicate any unplanned limitations to everybody involved so that they aren't surprised when they come to using your work.

☐ **9** **Provide a history** of how the work progressed.

- Write in pull requests and your logbook.
- Start with an overview and a summary of the problems your changes target. Then provide all of the details about what was changed, things that made the work easy/difficult, reasons behind change in approach, and anything else that would be useful.
- The people reading the pull request will include everybody from the discussions in step 1, interested colleagues, and often yourself in a few months time.

Material from steps further from step 5 is less formal, i.e. meeting minutes (step 1) and pull request descriptions (step 9). Material from the middle steps are increasingly formal, with source code being completely formal. The formallity is related to how much effort your colleagues want to see from you, i.e. "make the changes" is a small item in the middle with 4 steps on either side. From your colleagues' perspective, the most important thing is to see that you have put care and effort into understanding the problem and producing a well-considered solution.

You can, and should, ask your colleagues for assistance at any point that you don't understand something. However, make sure you take notes, and that they see you taking notes - they will likely be annoyed if they think you aren't learning from the assistance you requested.