Some Document Guidelines

Including Email, Code, and Reviews

Kees Goossens

v2.1, dated April 4, 2018

Ensure that your documents, presentations, emails, and source code conform to the following guidelines before you distribute them.

1 Professional Conduct

1. *Check the spelling* of your documents. Asking the reader to fix errors that can be trivially found is a discourtesy. Spelling Latex is possible with ispell, detex and spell, latex2html, WinEdit, and Word (which includes grammar checking), and so on.

2. Always use *page numbers* in your documents (including power point). If a paper must be submitted without page numbers, add them only in the draft versions. In presentations, it is annoying when a listener cannot refer to a slide number when asking questions. In papers, the order of the pages is not always obvious to the reader, and pages may be reordered during reading when referring to earlier or later sections.

3. For documents of more than a few pages a *table of contents* is very helpful for the reader. For large documents, such as theses the following lists should be added, where applicable: figures, terminology, abbreviations, symbols, definitions, theorems, etc. In all cases, add the page of the definition. An index can be very helpful, and is less work than might seem. Latex has packages to manage acronyms (`acronym`) and index (`makeidx`).

4. Use *version numbers* in all your documents (Latex or otherwise). Use filename_v*i*.suffix or year-month-day-filename.suffix or something similar. Make sure it *sorts in the right order* (i.e. older documents first) when listing a directory. Choose file names that are meaningful to others too.

   When returning an annotated document (e.g. with review comments or corrections) to the author, give it a different name, such as `filename_vi_your-initials.suffix\verb` (for example, `yearly_report_v1_KG\verb`). Do not increase the version number in this case.

   A well-organised European project defined their rules as follows:

```
<date of last modification>_<document title>_<version>_<partner>.<file extension>
<date of last modification>: YYYY-MM-DD, e.g., 2009-03-04
<document title>: Short but explicit title
<version>: Draft: v0-[x], e.g., v0-1, v0-12, Release: v1-0, v2-0, ...
<partner>: Acronym of partner [optional];
can be used to make clear who created the document /
made recent changes / added comments / etc.
<file extension>: doc, ppt, pdf, ...
Examples:
- 2009-03-04_INDEXYS_project-handbook_v1-0.pdf
- 2009-04-27_INDEXYS_deliverable-1-3_v0-3.doc
- 2009-05-12_INDEXYS_deliverable-1-3_v0-3_TTT.doc
```

The exact naming scheme is not too important, as long as you are consistent and you deal correctly with dates and versions.

5. When distributing different versions of a document, ensure that *comments on previous versions have been taken into account*. No-one likes re-correcting the same mistakes multiple times. Similarly, when an error has been pointed out in one place, check the entire document for the same and similar mistakes. Subsequent document versions should differ significantly: people do not have the time to proof-read a document in detail many times.

   If you do not agree with comments, and do not wish to implement them, then discuss this with the reviewer. Ignoring feedback tends to annoy reviewers (for a good reason), with unfavourable consequences.

6. Before submitting a paper, give co-authors sufficient time to review the paper. After submitting a paper, make sure that the git repository is up to date with the submitted pdf in the snapshot directory. Upon acceptance of the paper, repeat with the final version.

7. Clearly indicate when text or figures or data are reused from other documents. Without this, you commit (self)-*plagiarism*, which is not acceptable. Reusing text (e.g. introduction, related work, background) and pictures from your own papers in new papers is not to be recommended, as reviewers will spot this, and there is a high chance the paper will be rejected for this. (Rightly so, in my opinion.) This does not hold for extended journal versions of a paper.

8. Use *email Subjects* that reflect the purpose and content of the email.

9. Be polite in emails. No salutation, "hi," "howdy," "hey," and so on may be acceptable to your friends and direct colleagues. However, anything beyond that will need at least a "Hello FirstName," "Dear FirstName," or "Dear Mr./Mrs./Prof. SurName." Use "Dear Sir/Madam" when writing to a recipient of unknown gender. Err on the formal side when in doubt.

10. Include documents in your emails, rather than URLs to them. Not everyone is online when reading your email. Disk space is cheap. Check that you included the desired attachments.

2 Version Management (git)

Version management is desirable for many reasons:

1. You can go back to previous versions of your document or source code.
2. You can trace what changed in different versions of your document or source code.
3. You can work on the same document or source code from different computers at different locations (work, home, airplane, etc.).
4. Multiple people can work on the same document or source code. Conflicting changes are detected immediately by the version management system. This means that previous updates to code or text are not accidentally overwritten without any notice.
5. Multiple repositories act as remote backups. (In any case you should backup your files separately.)
6. You can make speculative changes, and either merge these with the source code if they work out, or else return easily to the original source code.

In the Electronic Systems group we use the git version management system, with the gitlab web front end.

1. *All* your source code should be version managed.
2. *All* your papers, technical notes, reports, and so on should be version managed.
3. If Microsoft word is used (unavoidable for many European projects), then use Track Changes and Comments whenever you modify the document. In any case, these multiple versions should be stored in our local git.
4. Commit frequently. It minimises the risk of losing work. It also minimises divergence from the main branch, where other people commit their work. This in turn minimises the effort to merge your changes back into the main branch (which is the end goal).
5. Commit whenever you have a new working version of your software (or good version of your document text), even if it took only 5 minutes. If during your software development, it stops working for an unclear reason, you can just go back to your last commit. This is much more efficient than repeatly using "undo" in your editor and rerunning until it works again.

6. Use short but useful description in version management comments. The empty string, "fixed a bug," and so on are not considered useful.

7. Each commit should be a self-contained change. It could be to implement one new feature, or to fix one bug. Multiple bugs or features should be checked in separately. This also helps with the previous bullet. It also allows (automated) backtracking to the smallest amount of code that caused an error.

8. Only check in source files. Do not check in generated files, unless there is a very good reason to do so.

9. For text files such as Latex it is preferable to not have very long lines. In particular, if a paragraph is written entirely on one line of text, then any change in the paragraph, however minor, will result in the entire paragraph being marked as changed. This is unhelpful to track changes. For example, what changed in the commit below (soft-wrapped to multiple lines to be able to make the point)?

```
-A data barrier is used to synchronize two tasks that have a data to
data dependency, that is, the sending task has a data-driven stop
condition and the receiving one has a data-driven start condition. The
sending task writes the output data, then updates the barrier and
blocks. The receiving task first updates the barrier and blocks
waiting for the sending task, and then reads the input data. In our
example, there are two data to data dependencies, from
t\textsubscript{3} to t\textsubscript{2} and from t\textsubscript{2}
to t\textsubscript{1} and the data barriers for them are shown in
Figure~\ref{fig:barrier} in green.
+A data barrier is used to synchronize two tasks that have a data to
data dependency, that is, the sending task has a data-driven stop
condition and the receiving one has a data-driven start condition. The
sending task writes the output data, then updates the barrier and
blocks. The receiving task first updates the barrier and blocks
waiting for the sending task, and then reads the input data. In our
example, there are two data to data dependencies, from
t\textsubscript{3} to t\textsubscript{2} and from t\textsubscript{2}
to t\textsubscript{1} and the data barriers for them are labelled
'data' in Figure~\ref{fig:barrier}.
```

   Preferably, each sentence starts on a new line: a change to a sentence then is then clearly shown as affecting only that sentence in the change log. As an intermediate, use hard line breaks (e.g. at column 70 or 80) inside paragraphs. This provides intermediate results. With this approach the difference between the two texts would have been shown as:

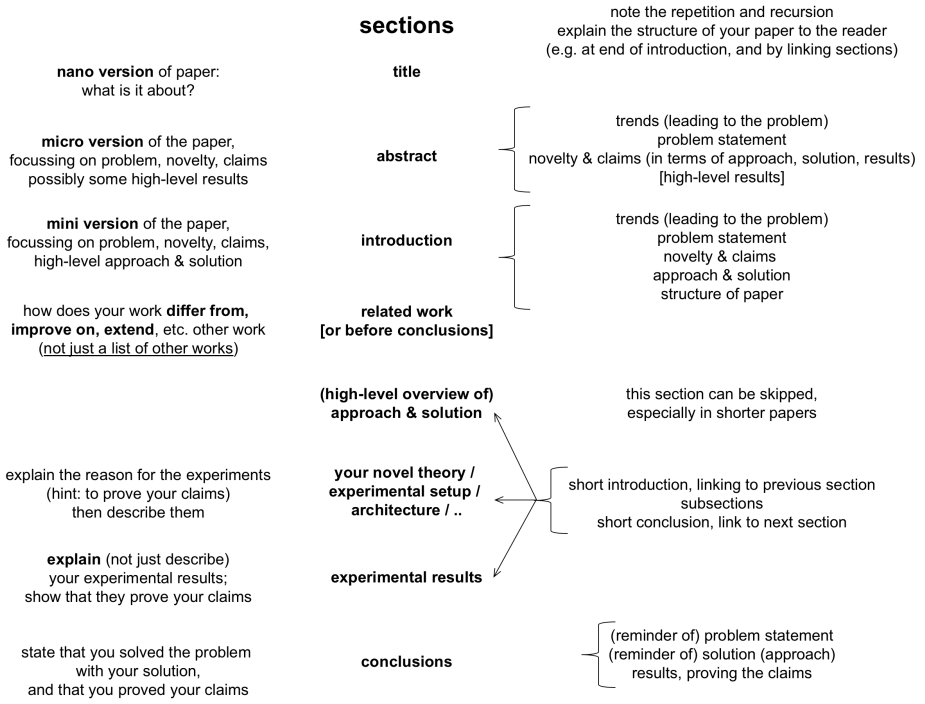
```
-to t\textsubscript{1} and the data barriers for them are shown in
```

```
-Figure~\ref{fig:barrier} in green.
+to t\textsubscript{1} and the data barriers for them are labelled
+‘data’ in Figure~\ref{fig:barrier}.
```

10. Branches are cheap. If you want to try something, but don't know yet if you want to keep it, then make a branch for it. Delete it if the experiment does not work out, or else merge it in your main branch.

3 Paper Structure

This diagram shows the general structure of a paper (or technical report). Note that the section on reviewing assumes that this structure is followed (or at least the information is presented in a similar manner). If this is not the case, then the paper is rejected.



4 English

Many and more of these recommendations can be found in The Elements of Style by Strunk and White.

1. Spell-check your text.

2. Use either British or American English consistently throughout your document(s).

3. It is important to write clear and unambiguous English. Because, since, thus, causes, follows (from), therefore, hence, but, (al)though, despite, however, while, moreover, etc. have different logical meanings. "A because B" ($B \Rightarrow A$) is not the same as "A thus B" ($A \Rightarrow B$), or "A though B." Be careful when using these words.

4. In the previous bullet, note the completely illogical, but grammatically correct, full stop (and comma and semicolon) *inside* the quotes.

5. Ensure that you use articles (a, an, the, every, etc.) before nouns when required (which is most of the time).

6. *Do not use "for verb-ing" but "to verb."* For example, "for explaining this point," should be "to explain this point." The "-ing" for denotes a continuous action, which is almost never intended.

7. Define your terminology, and use it consistently (preferably across documents). Without it, no description or discussion can be unambiguous. Do not use different words for the same thing or concept. Do not use the same word for different things.

8. Take care of singular and plural forms of verbs and nouns. "Each" is singular, "both," "all," and so on are plural. Consider, for example, "each level is important, but some levels are over-rated, and all levels are boring. Both cycle-accurate and transaction-accurate levels are described, and none are abstract. One or more levels are useful." Any can be used with both singular and plural verbs, depending on the context: "Any idea is welcome, if any of our problems are solved."

9. Write active sentences. Sentences and paragraphs should not be very long. Avoid the common phrases such as "the controller is orchestrating the SOC." It's shorter and better to write "the controller orchestrates the SOC." Another example is "x is used for initialising y," which is better written as "x is used to initialise y," or perhaps even "x initialises y."

10. That and which are different. The "thing that flies" means that flying is an integral part of it (or its definition). The "thing, which flies" is a thing that also happens to fly. That is not preceded by a comma, whereas which is.

11. The Oxford comma is the comma before the last and, or, and nor in a series of three or more. "A and B and C" is different from "A and B, and C," "A, B and C," and "A, B, and C." The last one uses the Oxford comma and is preferred, although the others may be used to indicate (subtle) subgrouping. Note that the comma is omitted with only two items: "A and B."

12. Either is always followed by (a single) or. Otherwise use multiple or. Same for neither - nor.

13. Note the difference between *it's* and *its*. The former means it is or it has, and the latter is the possessive adjective ("Our network is great, its name is Æthereal" or "it's called Æthereal").

14. "In order to" can almost always be shortened to "to."

15. Fewer and less are not the same. Fewer refers to a number ("this solution has fewer constraints"), less to a quantity ("this problem is less hard"). Similar cases are higher/lower (more/less than average height, "higher rate") and larger/smaller (a size that is more/less than normal, "smaller problem").

16. When discussing items, do so in the order in which they are introduced. See the previous item for an example.

17. Similarly, keep the legends (line type, marker type, etc.) of curves (experiments) that occur in different figures the same. They should be listed in the legend in the same order as they are discussed in the text. All axes on graphs should have a description of the quantity and units. Keep the ranges of the axes constant (as much as possible) over multiple graphs, to ease comparison. Verify that graphs are legible and understandable when printed in black and white. Graphs should not be misleading or exaggerating results.

18. The correct use of hyphens is not easy. They do make a difference in the meaning of sentences, however. Consider the example from Wikipedia: "more-important reasons," which means reasons that are more important, is distinguished from "more important reasons," meaning additional important reasons.

    The basic strategy, used by many, is to never use them. This is incorrect, but at least predictable, and many people will not mind (or notice). Correct hyphenation is preferred but can lead to ugly constructs, which can be avoided by rewriting the sentence. Examples:

    - "Power management:" power is the adjective of management; we manage power.
    - "Power-management strategy:" power management is the adjective, hence the strategy of how to manage power.
    - "Power management-strategy:" power is the adjective of a management strategy; presumably this means a management strategy that is powerful. You probably do not intend this. Note that without any hyphens, this meaning is implied.
    - "Dynamic-voltage-scaling method," meant to be parsed as ((dynamic (voltage scaling)) method), is an example of how multiple levels of adjectives lead to ugly hyphenation. Try to avoid these constructs.

Over time, hyphenated words have the tendency to lose the hyphen and be combined in a single word, e.g. data-base to database. Data flow, or data-flow, or dataflow: the choice is yours, as long as it is consistent, and preferably in line with the rest of the world.

19. Sub-sentences, such as this useless one, start *and end* with a comma (or other punctuation).

20. Avoid excessive use of abbreviations: it makes documents much harder to read for the uninitiated. Always introduce abbreviations on their first use. Preferably introduce them in the abstract, and in the introduction and conclusion sections, such that they can be read independently. If the use of non-obvious abbreviations is widely spaced in the document, do not use them, or reintroduce them after a long gap. For very long documents, such as theses, consider the use of the Latex acronym package.

21. When using an abbreviation such as SOC (System on Chip), you will often write either "a SOC" (pronouncing SOC as "sock" or "system on chip") or "an SOC" (pronouncing SOC as "es-o-see"). Either is fine, as long as it's consistent.

22. Do not cite papers in the abstract and conclusions; it should be possible to read them stand-alone.

23. Avoid colloquial expressions in written English, such as it's, can't, though, till; use it is, cannot, although, until, and so on instead.

24. Do not use turns of phrases like "it is obvious," "obviously," "as is well-known," and so on. Not everyone will find everything obvious, or will know certain things (even Moore's law). And if you are wrong on something "obvious," you will look not very smart. A reviewer may feel stupid and insulted and reject the paper.

25. Usually use one of "e.g." (meaning for example) and "etc." (meaning and so on). They can be used together, but it is not common. Place a comma before etc. when it is used with more than one item. Et al. is spelled as shown. "I.e." means id est, i.e. "that is."

26. Use "at an abstraction level," not "on a level."

27. "Whereas," "since," and "but" are not followed by a comma. Thus "Whereas, the other method fails." should be "Whereas the other method fails."

28. Do not use ellipses ("..."), especially instead of, or together with etc. and e.g. Do not use dashes ("–" or "—") for parenthetical sub-sentences; use commas instead. *Do* use them for ranges, especially in the bibtex `pages` entry.

5 Latex

Use Latex to write any significant ($\geq$ 1 page) text. Here are some common problems that you should address before distributing your text.

1. *Basic Latex hygiene.* You must be able to distribute the sources of the document as a single self-contained directory. No weird tools should be necessary for others to compile it. Do not use TEXINCLUDE and similar environment variables, which are system dependent. The tex files must be readable with a normal text editor (emacs, vi, nedit, whatever). It must compile without errors. A Latex document is a program, hence it should be clean, like any (C, C++, etc.) program. Preferably the directory should contain a makefile, with at least the targets `clean`, `all`. Figures are preferably in a subdirectory. Preferably use a single Latex file for all your text, rather than multiple `\input` files. This makes it easier to search for text, e.g. when making corrections.

2. Fix errors in the *bibliography and/or the bibtex file.* This includes forcing the correct capitalisation (e.g. SOC instead of soc) in titles. Use `"Left and Right {SOCs}"` or `{Put a {SOC} in it}`. Preferably use a common bibtex file for all your papers. Note: and then either copy it in the source directory (Kees's solution), or include the bibtex directory in the repository (Benny's solution). It may be smart to use a good bibtex key naming scheme, such as AuthorYearConference or similar, sort the bibtex entries in the bibtex file by key, and save the pdf of the paper using the same key, e.g. AuthorYearConference.pdf.

3. Citations have a *space* before them. Use the unbreakable space: `silly~\cite{key}` such that the citation is never detached from its preceding text. (Sub)sentences are separated by spaces too. Punctuation marks have no space before them, and one space after them. For example, "Hello , there" is incorrect, as is "Hello ,there." Latex introduces a (longer) inter-sentence space after "E.g.," "etc.," and "i.e.," which can be avoided by using a forced space after them: `e.g.\` This is not required when etc. occurs at the end of a sentence.

4. Use **bold face** very sparingly, and use *italics* with moderation. Never underline. When using italics in text, use `\emph{text}` or `{\it text}` or similar. The former is preferred, as it uses normal text when the text body is italic (such as often in the abstract). Do not use `$off$`: this looks horrible, as Latex will typeset the text in mathmode: off rather than *off*.

5. Similarly, in mathmode, when using text (i.e. two or more letters and digits) use `\textit{}`. For example, for ZAP_{eff}, use `$\textit{ZAP}_\textit{eff}$`, and not `${ZAP}_{eff}$`, which is typeset as ZAP_{eff}.

6. It may be smart to use the `\newcommand{}` command for recurring expressions like this. This also makes it easier to change notation. For example,

`\newcommand{\aethereal}{{\AE{}thereal}}`, which should be used as `text \aethereal{} text`. The point of this example is the `{}`, which inserts a space after Æthereal only when needed. Note the double curly brackets, they make the command more robust (it still fails in mathmode, though: see the previous example).

7. Units, such as nm, are preceded by a space (preferably unbreakable: `10~nm`) and are not italic.

8. Unbreakable spaces should be used also in conjunction with section, figure, table, and page number references: `as shown in Figure~\ref{fig:tradeoff}`. Note also the use of Latex labels that make sense, such as `\label{fig:tradeoff}`, `\label{sec:introduction}`, or `\label{item:abstraction}`.

9. Every figure must be referred to in the text. *Explain* a figure: why is it included, what does it intend to show, what should the reader learn from it, what conclusions can be drawn from it, etc.

10. Figures often save space in a document by first showing a reader what you mean, and then explaining the figure with some text. Describing with text a concept, design, architecture, application, graph, or other concept usually takes more space, and runs the risk that the user forms a different mental picture. A (technical) document with only text is harder to read (perhaps increasing the risk of rejection, in case of a paper).

11. When referring to figures, sections, tables, etc., with a number the word figure (etc.) must be capitalised, otherwise not. Page references are an exception. For example: "Figure 3 in Section 3.2 on page 6." Do not refer to figures (etc.) as "the figure below/above" and so on, but use number instead. Note that sub(sub)sections are referred to as sections, not subsections.

12. Figure captions are below the figure; table captions above the table. In Latex take care that your `\label{fig:text}` and `\label{tab:text}` are always below the `\caption`.

13. There is a difference between `‘ ’ " ‘‘` and `’’`. Always use `‘‘these quotes’’` to quote text "like this." Avoid single quotes, which usually indicates that the author cannot find the appropriate word (and might have benefited from the use of a thesaurus).

    Note that using quotes means that the text is literally cited, or a precise text, where every word counts and which must be read with extra care. Use quotes carefully therefore. Colloquial use, such as *all items on "sale,"* is confusing: why is the sale special, are the items really on sale, or is someone being sarcastic?

14. All your graphics should be scalable, and high resolution.

15. To save space in a paper, use a smaller itemsep (in itemised lists, including the bibliography). Use compactenum rather than enum, or inline the bullets by replacing by them by "a)" or "(1) or similar, in the text. Short bullets may be combined in a single sentence using a semicolon (;) between them. Otherwise use separate sentences.

16. Embed fonts in your pdfs.

17. Regarding presentation: use a *consistent style*, in terms of fonts, placement of text, headers and footers, colours, arrow types, and so on. This can be easily implemented by using a template. If you use OpenOffice or similar, distribute your presentations in the Powerpoint format. This allows easier re-use by others.

6 General Source Code

1. The main constraint on source code is *maintainability*. Write clean, structured code; e.g. use consistent indentation, capitalisation, commenting. Source code has a long life time, and will be read and modified by many people. You may not be available to explain it or fix it. Write simple code that is easy to understand, and hence easy to maintain.

2. *No program is finished until it has been comprehensively tested.* You are not allowed to utter phrases such as "it works, I only have to test it." Testing means demonstrating that the program works *correctly for all inputs*, not that there exists a single input for which the program seems to exhibit a correct output.

3. Compile with the strictest options, e.g. `-Wall -ansi` for C/C++.

4. All *warnings must be resolved*, for the simple reason that a) users when they compile your (generated) code do not look at warnings, and b) if they do, it is not clear what they should do (ignore it or fix it). Code should compile without a warning, or not compile, or fail with a run-time assertion.

5. Whenever you can, insert *assertions* to check properties that should hold. In C/C++ this can be at the start and end of functions, in VHDL this can be in the entity declaration (e.g. on static constraints between generics) or architectures (e.g. on dynamic constraints on signals and variables).

6. Use descriptive English names for variables, functions, methods, classes, files etc. even when it's not your native language.

7. Use makefiles where possible: a) to document the compilation parameters, and b) to ease the installation and reproduction of results for others.

7 Reviewing documents

Learn these three ways to read and review a paper:

1. 3-5 minutes
    - **read only the title, abstract, and conclusions; look at the references**
    - good, structured English? do know what the paper is about? whats the problem? is it relevant? has it been solved? is relevant work cited?
    - reject the paper if not meeting any of these
    - most people will read *your* paper like this. Make sure that you convince them!
2. 10-15 minutes
    - **also look at the figures in the high-level approach / solution, and (the figures & graphs in) the results section**
    - as above + do you have a high-level understanding of the solution? do the experiments make sense? have the claims been proven?
    - reject the paper if not meeting any of these
3. 1+ hour
    - **read everything, understand everything**
    - almost no-one will do this
    - reject the paper if you find real faults

The reasons for doing this are:

1. this "fail-fast" strategy saves you time
    - you can reject bad papers very quickly
    - no need to read bad papers in detail
    - you can spend more time on the interesting papers
    - or, if youre reviewing to accept or reject a paper, then your time should be spend on making the right decision for the borderline papers, not the bad or good ones (which are both easy to decide)
2. a paper that looks messy, or has bad English most likely has equally bad claims, solutions, and experiments; If the writers didn't bother with the details you can't trust the solution either. Details matter!
3. try to look at your own papers in the same way

Here is a list of basic things to check when reviewing a document:

- Do title, abstract, introduction, and conclusions match?
- Are the claims in the abstract & introduction justified, i.e. are they proven? This can usually be checked very quickly by reading the conclusions or the final parts of the results section.
- Is the problem defined well?
- Are the solution (and its concepts) defined well? Are the limitations and constraints defined clearly and up front?
- Is the paper well-structured in terms of sections, layout, figures, English? Is it understandable?
- Is there a related work section? Is it correct and fair to the other papers? Are all related papers that you know cited? Selective or biased citing is not allowed. Does it *compare the cited research to the paper* (and not just a description of what the related work does)?
- Are the experiments defined in sufficient detail (could you reproduce them based on the paper & the references)?
- Are the experiments presented well (graphs, etc. understandable)?
- Are the conclusions that are drawn from the experiments justified?
- What is the novelty of the paper? Give bonus points to cool ideas. Has it been done a million times before? Does it add anything to existing papers? If not, reject.
- Regarding expertise level: if you're doing your thesis on the topic, you're an expert. Otherwise you should probably fill in a lower level.
- You can write the review as a list of bullets (my style, less nice), or as running text (most other people's style, recommended, nicer). In the executive summary (e.g. comments to the Technical Programme Committee) write a short summary (at most 5 lines/bullets) with your recommendation and reasons for it.

More general guidelines:

1. Do not abuse your anonymity: *be polite*, and do not offend the authors. Imagine that it is your own paper that you are reviewing, and that you will receive the review.
2. A review is a document. Hence all of the above rules apply. Complaining about English or bad structure in a badly structured review with spelling errors is disrespectful.

3. Make the review as impersonal as possible: do not write "I do not understand this," "I would recommend," but e.g. "the text in paragraph I.B is difficult to understand," "please add more information on ...," "consider adding ..." and so on. If possible, give suggestions for improvement.

4. Review always to the same standard. It makes no difference whether you review for a top journal or a work-in-progress session of a workshop. It is up to the technical programme committee to decide what the minimum score is for paper acceptance.

5. When you submit a revised version of a document for a new review, include a cover letter that a) thanks the reviewers and editor for the good work, b) responds to each and every comment in detail. Try to make sure that each response results in visible changes in the document.

6. Apply a fail-fast methodology to reviewing. When a paper is really bad, then this is easy to write down in a short review. Do not read the details when it is clear that the paper makes no sense at a high level. Really bad documents are the easiest and quickest to review. Really good documents will generally take longer, because you do have to read the details, but writing the review will generally be easy. When reviewing multiple papers expect and aim to spend most of your time on decide on middle-quality papers: should they be accepted or not. Deciding this, and writing a good review takes time.

7. Reviewing papers should be useful for you: a) technically (the content of the paper, they way the problem is defined, approached, solved, etc.), and b) help you improve writing your own papers by understanding the review process.

8 Decyphering Kees's Written Feedback

Some symbols and abbreviations that you may find in the review of your papers:

- ✓: ok
- ?: unclear; the larger the question mark, the bigger the problem
- $\times$, $\ltimes$, etc.: not ok; the larger the cross, the bigger the problem
- $\lightning$: inconsistent, incorrect
- ˜: ok-ish
- large ˜ between two words or part of sentence: swap these two parts
- \\: break into two paragraphs or into two lines (e.g. in title)
- $\smile$: join two parts (paragraphs, sentences)

- $\Rightarrow$: implies
- $\not\Rightarrow$: non sequitur: does not follow
- $\therefore$ therefore
- $\because$ because
- $\circledast$, ①, ②, etc.: this text or remark is referred to later/elsewhere in the review
- o/w: otherwise
- w/o: without
- w/: with
- p: page, pp: pages
- def: definition, (un)def'd: (un)defined
- emph, it: emphasise, italicise
- ip, op: input, output
- (dis)cts: (dis)continuous
- req^s: request(s) / requirement(s), resp^s: response(s)
- trans: transaction
- beh: behaviour
- eqn: equation
- etc.