Electrical and Computer Engineering

Technical Documentation Guidelines

Philippe Kruchten
pbk@ece.ubc.ca

Version 1.3
January 28, 2019

Summary:
This document provides a set of general guidelines for the creation, evolution and maintenance of technical documentation for UBC ECE capstone projects. It focuses primarily on three key documents, applicable to a wide range of engineering project types: Requirements, Design and Validation.

Change history:

version	***by***	***change***	***location***
1.3 (Jan.28, 2019)	PBK	Misc. improvements proposed by R. Kerr	all
1.2 (Nov. 22, 2018)	PBK	Expand Design document	1.2
1.1 (Nov 19, 2018)	PBK	Misc. clarifications; added a summary section	5.0
1.0 (Nov 14, 2018)	PBK	Document created	-

Table of contents

1. Documents

A document is a piece of written, printed, or electronic matter that provides information or evidence or that serves as an official record.

What you are reading is a *document*: it provides information and guidance for the authors of the technical documentation associated with a given system or product under development.

Three keys technical documents are crucial to most systems or product sunder development:

1. A requirements document
2. A design document
3. A validation document

Depending on the nature of the product or system under development other documents may be useful or required. See Fig. 1.

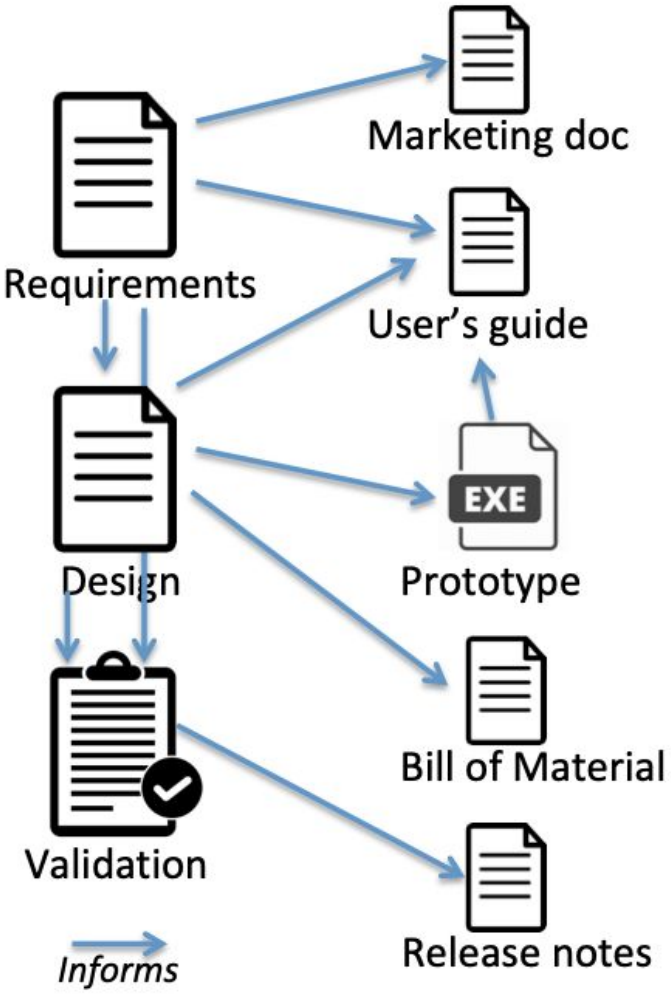

Figure 1: A few key documents for a computer-based product

1.1. Requirements document

This document captures information about *what the system or product does*; its function(s), or functionality, or capabilities, or services it offers, as well as information about quality attributes, such as its performance or dimensions or interoperability with other systems. The specification document may carry different names: Requirements specification, Software requirements specification (SRS), Product Vision, System Specification, Specifications,…

1.2. Design document

To design is to make choices or decisions about *how the system or product is built*: what it is made off (parts, subsystems), how they interface and interact, what tools are used to construct it. The Design document captures these design choices, as well as the rationale (or justification) of important choices, often relating them to the specification.

The design document provides an overview of the system needed by other engineers, maintainers, testers, etc. They will use this document as a roadmap to understand the system as a whole, and to locate components or areas of interest to them, then proceed to other detailed sources (e.g., source code, PCB layout) for very detailed information.

The design document must explain the system structure, operational and other views before going down into detail about the components. It identifies the interfaces and interactions between major components / subsystems / external services. Assuming that the reader has understood the information concerning functionality (included in the requirements specification) the design document should also address which component is “responsible” for what functionality. (Note that a Traceability Matrix helps to convey this info in very concise form).

1.3. Validation document

This document explains how the system or product under development, or some prototype of it, satisfies the *Requirements document*. The validation document assesses how functional and non-functional requirements are met --or not met-- and constraints satisfied, providing both the method of assessment (test, inspection, formal reasoning, experimentation, modeling and simulation, etc.) and the result of the assessment for a given version of the product.

To summarize:

- Requirements document: **what** the system does
- Design document: **how** is the system built
- Validation document: **evidence** that the system built does what the specification says

1.4. Audience

Write with the document's audience in mind.

Documents are consumed by other parties, or agents, or workers within the development organization or outside the development organization. It is important for the authors of the technical documents to identify who these other parties are, what they expect, what they know already, and what they need to know from the document.

For example, the requirements document will be used:

- By the *designers* and *constructors* of the system (internally)
- By *marketing staff* who describe the product to its potential market
- By other *designers*, of systems that have to interface with the product
- By *documentation people*, who produce users' guides (sets of instructions)
- By *external agencies* who may have to certify compliance of the product to some regulation or constraints
- By *executives and managers* who will drive the project(s) to develop the system
- If the product is provided under some contractual arrangement, the requirements document may be an integral part of the contract.
- Etc.

But in contrast, it is rarely the case that marketing staff or company executives have to deal with the *design document* or the *validation document*, or if they do, they would need some abstracted level.

To set expectations clearly, you may have to explain in the introduction section of the document who is the intended audience, and any prerequisites to understand the document.

2. Document timeline

Technical documents evolve over time. They are not written once, published and forgotten. In this respect, they are very different from a scientific paper or a thesis. They have a history of successive *versions* that follows and tracks roughly the history of the product or system they inform, which has versions, internal to the development organization, or visible to the end user.. The Apple Watch series 4 came 2 years after the Apple Watch Series 3, etc. The corresponding specification evolved. Software releases tend to have a smaller granularity (i.e, higher frequency) than hardware or complicated system releases.

It is important to:

- Associate the technical document to a given version or release of the product
- Keep track of significant changes from one version to another
- Keep a historical trail of document versions to assist in difficult situation to understand the evolution, or go back.

Associate the technical document to a given version or release of the product.

So clearly identify in the document for which release (internal or external) of the product the technical document is accurate:

Requirement specification for Product X - Release 10.2

Keep track of the change history.

And --keeping your audience in mind-- make it easy for them to identify what changes (addition, suppression, clarification, modification) you have made from a previous version. This is most often done by having a *Change History* table or *Change Log* at the very beginning of your document. The larger the document, the more important it is to have a concise and accurate “change log”. Let us assume that your *Requirement specification* is now 42 pages long. And you refine a function in section 4.5 page 37. The persons in charge of the web marketing material or the *Users’ Manual* do not want to have to scan the whole new document and compare it line by line with an older version, to find out if their work is affected or not by the changes you’ve just made. When receiving your new version, the change history is the place they will look first. You write it for them, not to justify the long hours you worked on making the change. Write with your audience in mind.

What to capture in a change history entry?

- Version number (optionally the date)
- Brief one or two line summary of the change
- Pointer to where the change actually is (that is, section numbers)
- (optionally) Author or initials of the author of the change or the person to contact about the change (we’ll discuss this more later; see section 4.1)

4.5 (May 10, 2018)	PBK	Added a cancel function on the new subscriber entry (screen change)	4.5.3
4.5 (Jan. 4, 2018)	JFH	Remove possibility of logging in via Google	5.2, annex 1
4.4 (Dec 3, 2017)		Temperature sampling rate: now 4 sec.	3.2, 6.4
...	...	...	

Organize the change history with the *most recent change on top*.

Very old entries, from 3 or 4 releases ago, or years ago, can be chopped off, to keep the change history manageable and usable (keep it around a page or 2 max).

Put your technical documents under version and configuration control, as you would do for software code, using tools such as Git.

3. Document structure

Define a structure for your document. Organize information by theme, or topics.

Make the structure of the document visible to the reader. You can achieve this in several ways:

- Use hierarchically numbered paragraphs, like the document you are currently reading: 1. 1.1. 1.1.1 etc.
- Insert a table of contents (ToC).
- Use larger fonts (and bold and italics) to make the headings stand out from the text in the body. Consider hanging headings (going into the left margin).

Do these steps when your document is over 5 or 6 pages long. It is pretty easy for the reader to understand the structure of a 5-page document, just by flipping through the pages. Beyond 5 or 6 pages, it is more convenient for the reader to go through a table of content to understand what information the document contains, and how it is organized, and then access directly the information she is looking for. Organize information logically, by topic, or by decreasing importance; alphabetical order is rarely the most logical (unless you introduce a glossary of terms).

Start with an overview of the document

Start the document with some introduction or overview: what is the content, the audience, and the structure. Add possibly a brief overview of the *product* if not readily available elsewhere. This will be useful for any newcomer to the project.

Defer very detailed information into annexes (or appendices)

Keep the level of detail consistent in a given section. Very detailed information on a given topic should be put in an appendix. Again, keep the reader in mind. If your document is handed over to a new hire, as part of the on-boarding process, what information would you like to give this new person on the team? You don't want to drown her with thousands of minute details, which would mean key important information is lost in the the forest.

Introduce a running header and/or footer.

For example:

Product XYZ Design document (version 4.5) *#page number#*

This again makes it easier for the reader having a pile of printed documents (or multiple windows on her screen) to rapidly identify which document she is reading). Use a font slightly smaller than the main body: if body is 11 points, use 9. Very, very long documents

(100 pages and more) may have running headers or footers *by sections*, again to ease access for the reader.

Define acronyms.

We techies love to use and create new acronyms. Some are well-known in our part of industry: WiFI, ADSL, TCP/IP, VLSI, FPGA ... Others are product- or company-specific. Again, to ease usage of the document by your intended audience:

- Make sure you fully spell out acronyms at the first place of use in the document, and if the document is long, spell it out again once in every major section.
- Have a table of acronyms in alphabetical order immediately after the table of contents.
- Minimize the use of acronyms to keep sentences easy to read and understand.

In some cases, you may also have to define a *glossary of terms*, if many terms you employ in the document are not used in the common sense found in a dictionary.

Capture references to other documents or sources

Technical documentation uses references to:

- Forward the reader to the original source of information and acknowledge other authors or copyright holders: give credit to whomever credit is due.
- Complement the document with information available elsewhere without having to explicitly copy it (DRY: Do not Repeat Yourself principle, or Single Point of Maintenance principle)
- Trace information upwards (back to the source or origin) ; for example trace between a decision decision and a specific requirement it addresses. More rarely, trace downwards.

Pick an easy to recognize citation style, and be consistent. You may use for example number between square brackets [2], pointing to a list of references (at the end, before any appendix). See IEEE citation guide in [6] for example.

[2] ISO/IEC 19505:2012 *Unified Modelling Language*, Geneva: ISO (2012)

Alternatively, you can use an APA style, with (Author, Year) citation [7].

Each reference should contain enough information for the reader to be able to retrieve the document you refer to. Use of URLs, or DOI (Digital Object Identifier) is very useful in that respect. Many references will also be to internal documents, belonging to the company.

Respect standard document structure.

There are circumstances, especially in contractual arrangements, where your document must follow some standard. For example, your requirements document must comply with IEEE Std 830-1998, I*EEE Recommended Practice for Software Requirements Specifications (SRS)* [1]. Keep the same headings and subheadings with their numbers. Simply write N/A

(= Not applicable) or TDB (= to be decided) for any section that is not relevant for your product or system. The reason is that your reader (for example, a government procuring agency) expects certain information to appear in well-defined places. For example, they would expect to find *Constraints* in section 2.4 of the SRS.

Limits font variety.

In your document layout design (or template), aim at clarity, not visual impact. Limit yourself to at most two fonts in a single document. Fonts with serif are easier to read in print, and then keep sans serif fonts for titles, headings, and table or figure captions. And again for legibility, avoid coloured fonts, grey fonts, or coloured backgrounds: make sure your text is well-contrasted; simple black-on-white works best.

4. Document style

4.1 Voice

Name your product or system.

Give an internal nickname or codename to your product. Pick something short and pronounceable. (Be careful also to pick a name that is not too controversial in any major languages.) Naming your product under development has multiple benefits:

- You avoid having to write or say “the product” or “ the system”, as we have done until now in this document. (By the way, we’ll be using “Schmoldu” as the name of the product in our example examples from this point on.)
- You can limit the use of the heavy passive voice. Rather than writing “Framework XYZ has been chosen”, you can write “Schmoldu uses framework XYX.”

In some cases the name you use internally maybe the same as the one used externally for commercial purposes. But it does not have to be. Early in my career I worked on telecom switches called internally in the labs System Y, and System Z, or just Y or Z for short, while the commercial names were Alcatel 2505, and Alcatel 2600, respectively.

Use “we” rather than “I”.

Most significant products are the work of a whole team. While you may be the person in charge of writing or updating the design document, use “we”.
Should technical documents be signed by their authors? In general no. They are the collective ownership of the company who produced them or paid for them; they are reviewed, corrected, approved by various people besides the person holding her hands over the keyboard. But for very practical reasons, it may be good to know whom to contact for questions, correction, evolutions, updates. The same goes for identifying the person responsible for individual changes in the change history. The idea is just to ease the

document and product evolution process, not to assign responsibility or blame, or cookie points to specific individuals.

Be plain, direct, factual, specific and pragmatic.

This is not literature, you are not writing a novel or an opera. No need to be lyrical: “After long hours of struggles and deliberation at the Barber study center, our team came unanimously (except for the obstinate opposition of Joe) with the bold decision to use the MEAN stack…”. Replace this by “Schmoldu is built on the MEAN stack (Mongo DB, Explorer.js, Angular.js, Node.js).”

Avoid gratuitously flowery language, full of unqualified adjectives or adverbs. Large, small, fast, user-friendly, efficient does not mean much if you cannot qualify these words, by associating some specific language that constrains them, usually something *measurable*.

“Our novel product greatly improves users efficiency, though greatly improved and effective user-interface.” Drop novel. Specify which class of user. Define “improvement in efficiency” with quantifiable metrics. Contrast with “Schmoldu release 5 provides a user-interface that improves data-entry time by 25% over release 5 for users familiar with the system, and accuracy by 7% on average (range 0% to 31%).”

Give ranges of acceptable target value rather than absolute numbers. Do not give unnecessary (and most often meaningless) decimal digits. Indicate the unit. “The switch operates when the temperature reaches 37.7778.” If your sensor is only accurate to 0.1 degrees Celsius, you should say something like “in the range 37.7 to 37.9 degrees Celsius”. (100℉ = 37.7778℃)

Strive for the six Cs: Consideration, Clarity, Conciseness, Coherence, Correctness and Confidence.

Explain why.

When you state something, such as an imperative but somewhat confusing requirement, or a bold design choice, explain *why*. Give the *rationale* for what you just stated (unless it is really obvious.). This rationale (or justification, or reason) can take many forms: by tracing to another piece of information, or by explaining the line of reasoning, or by contrasting alternatives that you have considered, and criteria for making a choice. Keeping a rationale in the document will foster a better understanding by your audience, and also provide some basis for evolution of the system: if some of the context changes then some earlier choices can be challenged and revised.

Example: Schmoldu uses React Native. This choice satisfies the requirement of portability across platform: iOS and Android, primarily. React Native is a path more widely used and more efficient than Xamarin or Flutter, which we have also evaluated.

Use lists and tables.

Rather than long-winded paragraphs, use lists.
Use bulleted lists when the order or sequence is irrelevant. Use numbered lists when the order is actually significant. Make lists with elements of the same grammatical form:

- All gerund
- All verbs
- All nouns … etc.

When items of your list have a regular structure, and contain 2 or more pieces of information each, consider using a *table*. For an example of table, see the Change history on the 1st page of this document. Tables are also good to present an evaluation of alternatives based on multiple criteria. This is a case where colour may help the reader navigating the message of the table; red, orange, green, for example.

4.2 Grammar

Use present tense.

In the past it was the tradition to use the future tense in requirements specification and even design documents. “The system will act as…”, “The product shall do…”. As your documents are living documents, and are associated with a specific version or release of the product, it is preferable to just use the present tense (even if what you are describing does not yet exist at the day and time you write it). By the time you have actual readers, the product will exist.

You may however use past tense or even future in some side notes or explanation. For example: “Note that we discontinued google login by release 3.1.” or “the multi-level undo function will not be activated until release 5, because too few users in our current installed base are using MacOS 10.x and later.”

Avoid passive voice.

When the verb is an action verb, you want to make it clear who is the actor, who is performing the action. The user enters the data. The administrator creates an account.
Or the designer (you) makes a design choice.
There are a few exceptions to this guideline: The actor is really unknown. You deliberately want not to name the actor. Or your are stating some very general truth. “Sustainability has been a constant concern throughout development.”

Break down very long sentences.

Break sentences that are over 3 or 4 lines into a sequence of sentences related to each other, possibly with “glue words” such as: “for example”, “similarly”, “however”, “ because of this”, etc.

Be gender neutral.

In English, an absence of pronoun refers to both genders. You can also alternate *he* and *she* (as we do here), though many find this awkward. You may also rewrite the sentences using *plural* to go around the issue of gender. *Users … they do this…* (This is harder to achieve in some languages than others; use of the plural does not work in French, for example.)

Use a spell checker.

But check what the spell-checkers suggest; they are not 100% right.

4.3 Illustrations

A judicious choice of *illustrations*, such a diagrams, screenshots, or photographs, will greatly improve understanding your document, from the perspective of the audience.

Use diagrams.

A good *diagram* can replace lengthy descriptions, especially in a Design document, and in particular to express complex relationships between elements. Make sure however that there is enough text associated with the diagram to introduce and explain the diagram. Keep text and figure as close together as you can. If you have more than 3 or 4, label your illustrations, and refer to them “as we shown in fig. 1, page 4”.
Be aware however, that good diagrams are harder to create and much harder to maintain than plain text. It is also difficult to express small variations and evolution in diagrams when going from one version of the document to the next.

Use a recognized notation for diagrams (or fully explain your conventions).

It is better to use a standard notation for diagrams, again in the interest of easing consumption of your document by your audience. An electronic diagram uses standard notation for transistors, capacitors etc, that do not need explanation. Do not create your own notation. Similarly for diagrams representing software, use recognized, standard notation like UML (Unified Modelling Language) [2], or IDEF0 [3].

In general if you have some “box-and-arrows” diagram, explain what a box mean, and what an arrow means. If you have boxes of various shapes and colours, explain the difference between them, with a *legend* attached to the figure, or some text in the body of the document. Do not gratuitously overload the reader with irrelevant information in your diagrams. Minimize meaning associated with colour. Some readers may be colour-blind, and sometimes documents are printed in black-and-white.

5. Summary

Write with the document's audience in mind.
Define a structure for your document.
Organize information by theme, or topics.
Associate the technical document to a given version or release of the product.
Keep track of the change history.
Start with an overview of the document.
Defer very detailed information into annexes or appendices.
Introduce a running header and/or footer.
Define acronyms.
Capture references to other documents or sources,
Respect standard document structure.
Limits font variety.
Name your product or system.
Use "we" rather than "I".
Be plain, direct, factual, specific and pragmatic.
Explain why.
Use lists and tables.
Use present tense.
Avoid passive voice.
Break down very long sentences.
Be gender neutral.
Use a spell checker.
Use diagrams.
Use a recognized notation for diagrams (or explain your conventions).

References

1. IEEE Standard 830-1998, *IEEE Recommended Practice for Software Requirements Specifications* (1998).
2. ISO/IEC standard 19505:2012 *Unified Modelling Language*, Geneva: ISO (2012).
3. ISO/IEC/IEEE standard 31320:2012 *Syntax and Semantics for IDEF0*, Geneva: ISO (2012).
4. W. Strunk, & E. B. White, *The Elements of Style*, 4th ed., Longman (1999).
5. *Chicago Manual of Style*, 17th ed., University of Chicago Press (2017).
6. *IEEE Citation Style Guide*, http://www.ijssst.info/info/IEEE-Citation-StyleGuide.pdf.
7. *American Psychological Association Publication Manual*, 6th ed., APA, Washington, DC, USA, 272p. (2010).