Documenting SW projects

One approach – Use this one, or define and follow a better one

31.10.2022

Haaga-Helia

Juhani Välimäki

Principles or goals

- Maintainable
- Generate what you can generate automatically
- Only to the needed level
- Avoid documentation that can be made unnecessary by other means
- School methods are often different ones, as they often form part of learning process
- In real project add only that documentation and visualization that is necessary
- Link when information is available elsewhere
- Provide table of contents where each developer can find just the interesting parts
  - Divide the content to shorter modules for easier learning but also for selection based on need and interest
- Optimize understanding and project reading speed, not project writing speed

Parts of project documentation

- Environment and project information
- Architecture introduction
- Database design and visualization
- Program code comments
- API documentation

Environment and project information

- **Hand-over** is importantant in all projects. We never know who might continue with the project
- README.md is our project de-facto standard.
  - Some Markdown markup examples given on the course. Check it out. Test whether works on Github pages.
- Make your project installation and configuration clear to the reader. An average IT professional has to be able to setup everything without further assistance!
- Don’t write redundant information. Thus no instructions on how to e.g. install Docker. Just list it as pre-requisites and possibly give link to elsewhere.
- Remember to explain the gitignored secrets config! (But no real values to the git repo (history)!)
- Be modular in your explanations, link to to other .md files in the project.

Architecture introduction

- Give just the big picture, put the reader on the map
- It’s a lot easier to study the project folders and code when you have some kind of idea what to look for
- Maybe some rough visualization of the architecture and very brief explanations of each part?
- Possible just in this level: Frontend: React, MaterialUI, AJAX with Axios, react-router-dom (v6 routig contexts used).
    - Would it be possible to link to e.g. the library list in package.json of a Node project?
- Keep this simple and as short as possible. And so generic that there should not be much need for changes later.

Database design and visualization

- In school you have learned good long processes for database design. From conceptual level ER diagrams, to logical level design, normalization, database diagrams, etc.
- Those are to some extent for learning the database design
- Some developers just do the database design and implementation at once (database diagram or just SQL DDL scripts). This of course requires some expertise and experience.
- Many tools offer generation of diagrams based on SQL DML Create table statements.
    - E.g. DBeaver offers adding more diagrams to the project and selecting which tables you want to include in there. DBeaver calls them ER diagrams, but they are actually **logical level *database diagrams***, table diagrams.
- In addition to generated database diagrams we need some data dictionary for:
    - a) avoiding using lot of aliases in project documentation, code and UI
    - b) agreeing on the units/limits etc.
    - c) general understanding of some complicated business case concept.
- Many databases offer the COMMENT ON feature of the SQL standard. Comments on tables and columns.
- Then we could avoid having separate database documents? All generated from scripts?

Program code comments

- First rule: Avoid need for code comments. Instead try to make your code clear with naming conventions and folder structure
  - Folder structure
  - Naming: Folders, files, classes, modules, functions, variables, attributes of objects
- Then, if still needed, only explain the confusing, irregular/unconventional/ or complicated parts
- Less is more. Quality over quantity. Think from reader’s point of view and starting point, not yours.
- Sometimes writing longer code helps, optimize reading speed, never writing speed.
- E.g. changing from the a ? b : c ternary operator to if-else might help the readability of the code and e.g. allow using explanatory variable names and comments next to lines

API documentation

- Libraries exist for making API documentation based on the API code
- We just need to add possible commentation as some kind of annotation or javadoc-kind of comments
  - (Javadoc: Write comments on certain style and they go to the javadoc tool generated HTML etc. Documentation)
- Thus, maybe use a library instead of non-updating Word document.
  - Didn’t we agree on this presentation mostly that we can almost totally remove non-generated, non-code or script linked documentation?