Principles of Software Design
Documentation, Maintenance, Logging

Robert Lukoťka

`lukotka@dcs.fmph.uniba.sk`

`www.dcs.fmph.uniba.sk/~lukotka`

M-255

Documentation in SW project

Documentation in software engineering is the umbrella term that encompasses all written documents and materials dealing with a software product's development and use. [1]

Why we need documentation?

Main purpose of documentation:

- To convey information when face to face communication not possible. E.g.:
  - Face to face communication between you now and you a year later is not possible :).
  - You have a meeting with the customer, and you feel it is inappropriate to repeatedly ask the same questions so you just make notes.
  - Make the future work faster/better.
  - To capture an agreement in a more binding way.
  - . . .

Any documentation you create should have a purpose, a reason for its existence.

Software documentation types

- Process documentation
- Product documentation
  - System documentation
  - User documentation

Process documentation

Examples:

- Meeting minutes
- Estimations
- Plans
- Company standards
- . . .

In many cases, process documentation is relevant only for limited time (until the process finishes).

System documentation

Describes the system and its parts [1]:

- Requirements
- Architecture, design
- Source code documentation
- UX (user experience) design documents
- Quality assurance documentation (testing strategy, plan, the actual tests, ...)
- Help and maintenance
- ...

User documentation

How to use the system:

- FAQ
- Beginner's guide (contains the usual use cases)
- Full user manual
- Manual for system administrators
- . . .

Various actors may interact with the system differently, they may need specialized manuals.

Good practices

- Just barely good enough
- Everybody contributes to the documentation.
- Having out of date documentation may be worse than having none, but slightly out of date documentation may still be quite useful (especially if you are aware of the fact).
- . . .

Good practices - Agile/Lean Documentation [2]

Writing

- Prefer executable specifications over static documents
- Document stable concepts, not speculative ideas
- Generate system documentation

Simplification

- Keep documentation just simple enough, but not too simple
- Write the fewest documents with least overlap
- Put the information in the most appropriate place
- Display information publicly

Good practices - Agile/Lean Documentation [2]

Determining What to Document

- Document with a purpose
- Focus on the needs of the actual customers(s) of the document
- The customer determines sufficiency

Determining When to Document

- Iterate, iterate, iterate
- Find better ways to communicate
- Start with models you actually keep current
- Update only when it hurts

General

- Treat documentation like a requirement
- Require people to justify documentation requests
- Recognize that you need some documentation
- Get someone with writing experience

Maintenance

Maintenance is significant: it accounts for $40 - 80\%$ software lifetime cost [3].

Maintenance is not only about bug fixing [4]:

- Adaptive: Modifying the system to cope with environment changes (computer, OS, etc.) - 25% of work
- Perfective: Modifying the system to satisfy new or modified requirements - 50% of work.
- Corrective: Correcting discovered problems - 20% of work
- Preventive: Detecting and correcting latent faults before they become effective faults - 5% of work
- . . .

Although the studies are old, general consensus is that the ration of software maintenance has increased since then.

Maintanance

Maintenance is expensive: a feature is $2 - 100\times$ more expensive as in regular development. Why?

- Loss of knowledge:
  - employee turnover,
  - time (people forget stuff)
  - time (systems and technologies get outdated and thus developers are not as familiar with them as they used to be)

- Maintenance departments contain more junior developers on average.

- No incentive for the developer to offer low price.

- . . .

Factors affecting maintenance cost [5]

- Team stability
- Contractual responsibility
- Stuff skills
- Inadequate configuration management
- Inadequate documentation
- Inflexible design/architecture
- Program age and structure.

Logging

Why we need logging?

- How do we know about problems encountered in production?

Is logging architecturally significant?

- It affect all/most of the parts of the SW system $\rightarrow$ Yes.

Problems

What is hard:

- How to log in such a way that we do not have too many logs but we still have the data we need?
- Is logging flexible enough so one can change the configuration easily to obtain more detailed logs concerning something when needed?
- Logging code increases the total length of code, it makes e.g. business logic harder to read.
- How to log in libraries?
- . . .

We introduce Python logging library. It gives an up-to-date approach on how to deal with many of those issues.

Logging in Python

We have the following classes:

- Logger - object that receives the logs.
- Handler - decides what to do with the log, one logger can have multiple handlers.
- Filter
- Formater

Logging in Python

- Loggers are singleton objects and live in hierarchy corresponding to the hierarchy of packages. To use a logger within a unit we use:
  `logger = logging.getLogger(__name__)`
- This allow us to configure logging for each module separately.
- We can pass logs ho higher level loggers within the hierarchy, that is `root.a.b` logger can pass logs to `root.a` logger which can pass them to `root` logger. This allows us to set reasonable approach at the right level.

Logging in Python

- The levels of logging:
  - `logger.debug(...)` - Detailed information, typically of interest only when diagnosing problems.
  - `logger.info(...)` - Detailed information, typically of interest only when diagnosing problems
  - `logger.warning(...)` - An indication that something unexpected happened, or indicative of some problem in the near future.
  - `logger.error(...)` - Due to a more serious problem, the software has not been able to perform some function.
  - `logger.critical(...)` - A serious error, indicating that the program itself may be unable to continue running.

Logging in Python

Check here how does it work: Logging flow

Logging in Python

Some of Logger methods:

- *.propagate* - should the event be passed to higher level (ancestor) loggers?
- *.setLevel* - be careful that if something propagates to higher level logger, the level is not checked.
- *.addFilter*
- *.removeFilter*
- *.addHandler*
- *.removeHandler*

Logging in Python

Some of handler object methods

- *.setLevel*
- *.setFormatter*
- *.addFilter*
- *.removeFilter*
- *.flush*

Logging in Python

You do not need to write your own handlers, here are some that are available:

- StreamHandler
- FileHandler
- NullHandler
- RotatingFileHandler
- TimedRotatingFileHandler
- SocketHandler
- DatagramHandler
- SMTPHandler
- HTTPHandler

Resources I

- Agile/Lean documentation best practices
- Software maintenance - Wikipédia
- Logging Cookbook

References I

▤ Software Documentation Types and Best Practices

▤ Agile/Lean documentation best practices

▤ R. L. Glass, "Frequently forgotten fundamental facts about software engineering," in IEEE Software, vol. 18, no. 3, pp. 112-111, May-June 2001.

▤ Lientz B., Swanson E., 1980: Software Maintenance Management. Addison Wesley, Reading, MA

▤ J. Kostičová: Documentation & Maintenance