Maintenance of technical and user documentation

Article · April 2015

CITATIONS
2

READS
14,116

2 authors, including:

Mario Gastegger
TU Wien
6 PUBLICATIONS **2** CITATIONS

SEE PROFILE

Maintenance of technical and user documentation

Mario Gastegger 0726289

Simon Zünd 1025990

17th May 2015

Software maintenance is usually the biggest part of the software development life cycle. Studies have shown that up to 60% of software maintenance time is spent on understanding and comprehending the maintained software. As such, documentation maintenance and documentation in general deserves attention. Documentation can be divided into user documentation, targeted at end-users, and technical documentation, targeted at developers, maintainers, system administrators, etc.

1 Introduction

Documentation is about documents, which communicate information. Those documents provide information for and about a certain object, process or topic. Documentation can be published in digital (CD, DVD, bluray disc, memory stick, download, as a web site,...) or in analog (book, paper, poster, photo...) form. Further more, digital documentation can be presented in an interactive manner, which increases ease of use. Examples for interactive features are, among others, cross references, search, tip-of-the-day features, contextual information, wizards or knowledge based help systems.

Another important characteristic of documentation is the level of detail. There may exist several documents describing the same aspect, but are intended for a different audience (with varying domain knowledge) and thus serve a different purpose. Software system underlie continuous changes throughout their whole life cycle. As a consequence, also the amount of information grows continually. For international software systems, the documentation has to be available in multiple languages. All these aspects contribute to the problems of documentation: Keeping up with changes during the development, propagating changes in requirements documents, design documents to the development and keeping documents on the desired level of quality.

The above already indicates, that most documentation artifacts also underlie frequent and sometimes fundamental changes and the task of maintaining them require additional effort in personnel, time and thus money.

Documentation in context of software engineering can be defined as an artifact whose, purpose is to communicate information about the software system to which it belongs [7]. In this paper we distinguish two types of documentation. First, the technical documentation which includes all pre-release documentation. And second, user documentation, which includes documentation for installation, operation and management of software systems [1].

This paper focuses on task of maintaining technical and user documentation. Since it is important to understand the nature of technical and user documentation, section 2 will define and describe both terms in greater detail. Section 3 tries to answer some questions about documentation quality. Section 4 investigates the organizational measures to be taken to ensure proper creation and maintenance of documentation. Section 5 introduces and proposes environment settings, tools and procedures to support creation, maintenance and the use of documentation. section 6 concludes the paper with a summary.

2 About technical documentation and user documentation

In this section, we will define technical and user documentation and investigate both subjects from different perspectives. Understanding the different aspects of documentation helps to deduce requirements for maintenance.

2.1 Technical documentation

Among all the recommended practices in software engineering, software documentation has a special place. It is one of the oldest recommended practices and yet has been renowned for its absence. There is no end to the stories of software systems, especially legacy software, lacking documentation or with outdated documentation. For the last decade, the importance of technical documentation (and documentation in general), has been stressed by educators, software development processes, standards, etc. and despite of this, it is still discussed why documentation is not generally created and maintained. In recent years, this topic gained new interest (from [6]):

- Agile methods question the importance of documentation as a development aid
- The growing gap between *traditional* (e.g. COBOL) and up-to-date technologies (e.g. OO or web-oriented) increased the pressure to re-document legacy software.

Typical examples of technical software documentation include: requirement specifications, design (architectural) documents, source code comments, test documents and defect (bug) reports. The authors of [9] summarize benefits of technical documentation from other articles. Figure 1 shows the result. The benefits have been classified into four categories: (1) maintenance aid, (2) development aid, (3) management decision aid, and (4) other.

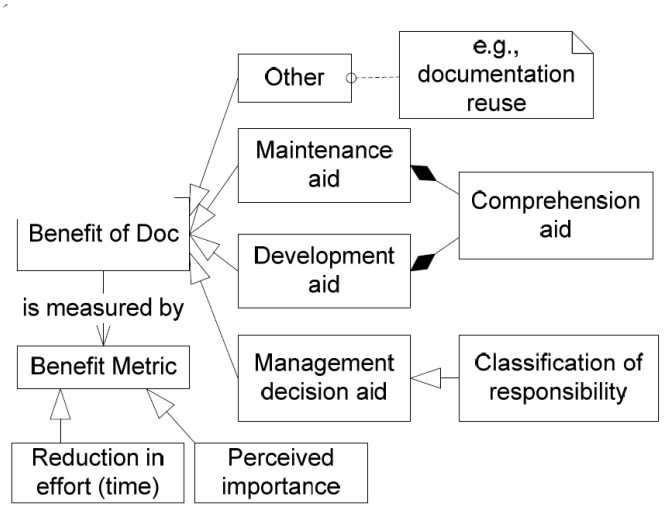

Figure 1: A meta-model for documentation benefit (from [9])

2.1.1 IEEE 155289

There exists an international standard, IEEE Std 15289 - Systems and software engineering - Content of life-cycle information products (documentation) [3], which describes its purpose as "to provide requirements for identifying and planning the specific information items (information products) to be developed and revised during systems and software life cycles and service processes". The IEEE standard 15289 is based on the life-cycle processes specified in ISO/IEC 12207:2008 and ISO/IEC 15288:2008, which are the current umbrellas containing most standards regarding software development. The intended users of the documentation standard are (non-exhaustive):

- **Project managers** responsible for the *Information Management process.*
- **Project managers** responsible for identifying document contents and documentation requirements, to help determine what should be documented, when the documentation should occur, and what the contents of the documents should be.
- **Acquirers** responsible for determining what documents are needed so they can ensure the quality of the delivered product.
- **Individuals** who write documentation.

The standard then describes in great detail different types of information items, such as, descriptions, plans, policies, procedures, reports, request, specifications, etc, how these information items should be structured, and most important, what life cycle phases of the software development process should produce which information items (and their types). The standard also provides procedures which help in identifying concrete information items on a per-project basis, as each project has usually different requirements regarding documentation.

2.1.2 Documentation as part of other standards

Usually the standards describing the different software life cycle phases also contain chapters with guidelines and requirements for information items. An example is the standard for requirements engineering (IEEE 29148) [4]. Section 7 - 9 of this standard describe in great detail the structure and content of resulting documents, such as:

- Stakeholder requirements specification document (StRS)
- System requirements specification document (SyRS)
- Software requirements specification document (SRS)

2.1.3 Technical documentation in the context of software maintenance

In [6] Souza et al conduct a study in search for documentation which is essential to software maintenance. They focus on software maintenance, as it is often the predominant activity in software engineering. Also the lack of up-to-date documentation is one of the main problems that affect software maintenance and has a big impact on time spent. [6] cites several studies which report that 40% to 60% of time spent in maintenance is actually studying and understanding software.

The study conducted in [6] consists of two surveys. The first asked maintainers what artifacts and information items they deemed important, the second asked what artifacts the maintainers actually used. The survey identified that the most important artifact is still the source code and comments. Although data models and requirement descriptions were deemed important as well. Surprisingly, and contrary to literature, architectural models and other general views of the system were not deemed as important. One reason could be that such views are only inspected once to get a general idea, but are not consulted in the future, resulting in a difference between quantity and quality of documentation.

Other studies come to different conclusions on what documentation they deem important:

- Tilley in [13] stresses the importance of architectural documentation
- In a workshop organized by Thomas and Tilley at SIG-Doc 2001, they state that "no one really knows what sort of documentation is truly useful to software engineers to aid system understanding"
- Cioch et al. [5] specifies four stages of experience: from new comer, first day at work; to expert, after some years. For each stage, they propose different documents. New-comers need a general overview whereas experts need low level documentation as well as requirement description, and design specifications.
- Ambler recommends documenting the design decisions, and a gneral view of the design: requirements, business rules, architecture, etc.

- Forward and Lethbridge [8], in their survey of managers and developers, found that specification documents to be the most consulted whereas quality and low level documents are the least consulted.

2.2 User documentation

Software user documentation is as old as software development itself. The actual users of software systems need information on how to interact with the software system and know how to accomplish their tasks efficiently. These aspects and the fact that documentation often varies in quality and structure have lead to research and also the development of standards around documentation. Standards for user documentation have been summarized first in [1], the current version is IEEE Std 1063-2001 [2].

The IEEE Standard for Software User Documentation [1] defines it as follows: User documentation guides users in installing, operating, and managing software of any size, and conducting those aspects of software maintenance that do not involve modification of the software source code.

The major consideration in user documentation are the audience, the purpose, the form and the format.

2.2.1 The audience

It is essential for user documentation to fit the user's needs. An audience analysis has to be done. To illustrate the importance, consider a software system used in home automation for the elderly and disabled. There is very little technical knowledge to be expected, which has to be considered in the conception of the documentation. Some examples of the vast variety of audiences are elderly people, disabled people (color blindness, deafness, ...), software admins, end-users (with domain knowledge), users without domain knowledge, support staff, sales personnel, marketing personnel, users of public APIs (web, plug-in,...)... These examples already implicate, that documentation can not fit all the needs and thus has to be specifically tailored for a handful of audience characteristics.

2.2.2 The Purpose

It is obviously important to capture the purpose of the software system. Furthermore the documentation has to capture the work flows which accomplish the various tasks of the domain and what to do in case of errors (troubleshooting). It should guide the user on the way of accomplishing certain tasks with the software system. Another approach is to explain the software system itself. A reference manual explains, among others, UI elements, menu structure, parameters, input data and output data.

2.2.3 The form

The form is closely related to the audience and the purpose. Documentation can occur in many different ways. The more traditional forms like manuals, printed reference

manuals and guides, which can also be provided in digital form. And the digital forms like screen casts, wizards, getting started (Introductory pages which explain the basic use on first application start), tutorials, quick tips (short explanation on every application start), contextual hints (hover hints or a link to the manual), knowledge-based help systems, manuals, 3rd party literature (books, articles), announcements ("news" items which provide information on changes in system parameters or on new features as mentioned by [12]). The digital forms can provide added value to the user through a search function, a linked index and interactivity in general. The implications of printed (versus digital) documents are obviously, the higher costs of printing and in case of changes or mistakes. Nowadays, in the end, all documents exist also in digital form. The documentation is created and maintained as digital files, which makes additions, changes and corrections easier traceable and more closely linked to the development process of the software system.

3 Documentation quality

During the software development life cycle, the quality of documentation artifacts developed in earlier phases, e.g. requirements, will generally have a big impact on the quality of artifacts developed in later phases.

3.1 Attributes

Similar to software quality, which has various attributes, quality of documentation has various attributes as well, such as: accuracy, completeness, consistency, correctness, information organization, format, readability, spelling and grammar, traceability, trustworthiness, and up-to-date-ness [14].

Quality of documentation has been analyzed in a large number of studies so far, which are based either on experts' opinion (i.e. using surveys), or by mining project data and using metrics to measure documentation quality [9].

4 Organizational aspects of the creation and maintenance of documentation

As the software system itself, also the documentation evolves. Documentation should be created and maintained in a process. The most important standards which describe the processes to manage documentation items are IEEE 12207:2008 and IEEE 15289:2011. This section aims to show the most important/critical aspects to ensure successful documentation creation and maintenance in the industry. Once again there should be made a distinction between technical and user documentation.

Writing

For technical documentation the writers and also the audience is usually the developers. Since the main focus is the communication of implementation details, they may not need special writing skills. For user documentation on the other hand, it is necessary for the writers to have didactic skills and knowledge about the audience specific language(in addition to domain knowledge and a high level of understanding of the software system and its use). As a result, the writers need specialized training.

Domain specifics

Since it is almost impossible to avoid any domain specific language in documentation, the whole project team has to be introduced to the domain. This has to be done in advance. It consumes time(workshops, training, meetings, ...) and personnel(the customer, internal and external experts) but is necessary to ensure a team wide domain knowledge.

Legal obligations

For certain domains (medical software, aerospace, ...), legal aspects and regulations, have to be respected. Therefore it is necessary for the legal department to acquire domain knowledge and knowledge about these legal obligations in advance.

Support

Support costs arise during the last phases of the life cycle of a software system. To reduce the costs for support during that possibly long period, the investment into more effective documentation pays off. Another way to reduce support costs is to build a community platform around the software system, where users provide support for other users.

Marketing

Technical and also user documentation is important for marketing, technical details may be the only or the most important distinctive features of the product.

Use cases can be transformed into scenarios, which transport a certain life style or just underline the ease of use.

Also the user documentation itself can transport a marketing message about the software system or the company [15].

5 Procedures, tools and environment

About choice of technology and special advantages and disadvantages for documentation.

5.1 Literate programming

The term literate programming was introduced by Donald Knuth [10] and refers to a concept where source code and documentation are the same. Thus, the creation and maintenance of documentation and the actual changes to the software system, are done at the same time. In literate programming, the syntax has to be flexible enough to allow the programmer to formulate their programs in natural or nearly natural language. Examples of language which encourage a literate writing style are cweb, forth, haskell, CoffeeScript.

5.2 Documentation generator (Generation and extraction)

Document generators are used to extract information from source code level objects and build structured documents of it.

This approach works best for technical documentation. There are two types, first the generators which need source level comments (javadoc, doxygen, robodoc, ...) and second, those which generate documentation from source code only [11].

The obvious advantage of documentation generators, is, when code changes, that documentation only needs to be regenerated and not rewritten by humans(assuming the developers also adjust the comments along).

The first type is already widely used. It's effectiveness depends heavily on the developers motivation, code documentation conventions, the existence of references or pre-conditions, post-conditions and invariants. This type is most effective for interface documentation or API documentation.

5.3 Maintaining documentation in CI environments

In recent years, continuous integration (CI) and continuous delivery are getting more and more popular. Originally designed to build, test and package software in regular intervals, CI environments can also be facilitated to keep documentation up-to-date or to check if the documentation and the project are out-of-sync. This is especially true for documentation which is generated via automated tools from the project source. Examples include source code documentation like JavaDoc or generated class / package diagrams.

5.3.1 Requirements as BDD test cases

Also in recent years, in tandem with the agile software development movement, a school of testing and developing emerged called *behaviour driven development (BDD)*. One aspect of BDD is, that requirements are specified in a structured way, which can then later be turned into automated acceptance or system tests. In this approach, the behaviour specification does not only serve as documentation, but is also tied to the automated test suite. One can now see, that in a CI environment, not only suitable documentation can be generated out of these specifications, but it is also easier to keep documentation

and actual software in sync. Examples include Cucumber and Jnario (Java). Figure 2 shows such a requirement described for Cucumber.

```
Feature: Refund item

  Scenario: Jeff returns a faulty microwave
    Given Jeff has bought a microwave for $100
    And he has a receipt
    When he returns the microwave
    Then Jeff should be refunded $100
```

Figure 2: Requirement description for a hypothetical refund feature

5.4 UML Use case diagram

Requirements in general and most of all user stories and use cases are a very good resource for user documentation. They state what the user wants to do and how he does it. One approach to keep documentation and software in sync would be to tie them together via use cases and tests. I.e. tests are tied to use cases or user stories and the user documentation of features is tied to the specific use cases.

5.5 Tools used in the OSS community

Developers of open source software (OSS) usually do not work in the same place, often not even in the same time zone. So communication and documentation in OSS is as important as in *regular* software projects. This section presents some tools which became popular in recent years and how they tie in with documentation maintenance.

5.5.1 Wikis

Documentation in wikis is often targeted at end-users. For libraries they contain installation and usage instructions for other developers. For programs they contain end-user documentation like GUI and typical work-flow descriptions. Depending on editing policies, the original developers are even able to delegate some of the documentation work to motivated end-users.

Some wikis even allow version tagging of articles and / or parts of articles. This means that for new software versions, articles have to be reviewed and checked for correctness and actuality before they receive the tag for the new version. Nowadays wikis are so popular, that code hosting sites like GitHub and Bitbucket provide wikis on a per-project basis.

5.5.2 Code hosting sites

Originally developed on top of version control systems like Git or Mercurial, code hosting sites provide more and more social and other project management features like wikis, artifact hosting, bug and issue reporting, etc. These systems allow for easier change documentation:

- Change logs can automatically be generated out of commit histories.
- Bugs and features can be tracked via version control as tickets can be referenced in commits and vice versa.
- Discussions regarding features and bugs are tied directly to the code via pull requests or bug reports.

Examples of such sites include GitHub and Bitbucket.

6 Conclusion

Although most standards, software development processes and models stress the importance of documentation, many projects shine in the lack there-of. Discussions about the correct amount of documentation is as dominant as discussions about how and what to document, or even to document at all. In this paper we tried to present some of the official standards and findings in research. We also tried to show how some of these findings are practically applied in the field.

References

[1] Ieee standard for software user documentation. *IEEE Std 1063-1987*, pages 1–20, Aug 1988.

[2] Ieee standard for software user documentation. *IEEE Std 1063-2001*, pages 1–24, Dec 2001.

[3] Systems and software engineering – content of life-cycle information products (documentation). *ISO/IEC/IEEE 15289:2011(E)*, pages 1–94, Nov 2011.

[4] Systems and software engineering – life cycle processes – requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, pages 1–94, Dec 2011.

[5] F.A. Cioch, M. Palazzolo, and S. Lohrer. A documentation suite for maintenance programmers. In *Software Maintenance 1996, Proceedings., International Conference on*, pages 286–295, Nov 1996.

[6] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting &Amp; Designing for Pervasive Information*, SIGDOC '05, pages 68–75. ACM, 2005.

[7] Andrew Forward. Software documentation – building and maintaining artefacts of communication. Technical report, 2002.

[8] Andrew Forward and Timothy C. Lethbridge. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering*, DocEng '02, pages 26–33, New York, NY, USA, 2002. ACM.

[9] Golara Garousi, Vahid Garousi, Mahmoud Moussavi, Guenther Ruhe, and Brian Smith. Evaluating usage and quality of technical software documentation: An empirical study. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, EASE '13, pages 24–35, New York, NY, USA, 2013. ACM.

[10] Donald E. Knuth. Literate programming. *THE COMPUTER JOURNAL*, 27:97–111, 1984.

[11] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker. Automatic generation of natural language summaries for java classes. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pages 23–32, May 2013.

[12] Jack Thompson and Susan G. Howell. User documentation: Types and preparation methods. In *Proceedings of the 5th Annual ACM SIGUCCS Conference on User Services*, SIGUCCS '77, pages 138–141, New York, NY, USA, 1977. ACM.

[13] Scott R. Tilley. Documenting-in-the-large vs. documenting-in-the-small. In *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Distributed Computing - Volume 2*, CASCON '93, pages 1083–1090. IBM Press, 1993.

[14] Marcello Visconti and CurtisR. Cook. Assessing the state of software documentation practices. In Frank Bomarius and Hajimu Iida, editors, *Product Focused Software Process Improvement*, volume 3009 of *Lecture Notes in Computer Science*, pages 485–496. Springer Berlin Heidelberg, 2004.

[15] N.J. Walters and C.E. Beck. A discourse analysis of software documentation: implications for the profession. *Professional Communication, IEEE Transactions on*, 35(3):156–167, Sep 1992.