UDC 004.01

V. Herwig, DSc.

DOCUMENTATION OF SOFTWARE SYSTEMS

Abstract. *With software being all around us, it is surprising how little the documentation of it has developed during the last decades. Software system documentation is however most critical during maintenance tasks, where tremendous amounts of time are spend understanding the software and its documentation, before the actual maintenance work can start. This paper gives a status of current documentation practices as well as a view on its quality. The purpose of this paper is to create a discussion about those practices to foster ideas for the future.*

Keywords: *information technology documentation, Source code documentation, In-line documentation, System knowledge*

Ф. Хервиг, д-р техн. наук

ДОКУМЕНТАЦИЯ СИСТЕМНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Аннотация. *В статье рассмотрена проблематика и пути решения недостаточного количества документации, создающейся для программного обеспечения в течение последних десятилетий. Документация для систем программного обеспечения является важным фактором при техническом обслуживании, когда огромное количество времени тратится в попытках понять программное обеспечение и его документацию до фактического старта работы. В статье рассмотрен статус существующей практики документирования, а также произведён обзор качества документации. Целью данной работы является создание обсуждения существующих практик документирования для содействия развёрнутой дискуссии в будущем.*

Ключевые слова: *информационные технологии, документация, документация исходного кода, эксплуатационная документация*

Ф. Хервіг, д-р техн. наук

ДОКУМЕНТАЦІЯ СИСТЕМНОГО ПРОГРАММНОГО ЗАБЕЗПЕЧЕННЯ

Анотація. *У статті розглянуто проблематику та шляхи вирішення проблеми недостатньої кількості документації, що створюється для програмного забезпечення протягом останніх десятиліть. Документація для систем програмного забезпечення є важливим фактором при технічному обслуговуванні, коли величезна кількість часу витрачається в спробах зрозуміти програмне забезпечення і його документацію до фактичного старту роботи. У статті розглянуто статус існуючої практики документування, а також проведений огляд якості документації. Метою даної роботи є створення обговорення існуючих практик документування для сприяння розгорнутої дискусії в майбутньому.*

Ключові слова: *інформаційні технології, документація, документація вихідного коду, експлуатаційна документація*

Introduction

When the United States government decided to change the health care system for their people, they realized that out of efficiency reasons, the citizens have to self-manage their data and in times of the internet it became clear, a website has to be developed for this purpose. It turned out to become one of the big software projects of the United States government (healthcare. gov) and unfortunately one of the biggest failures. Estimates are that only 1 % of people managed to successfully enroll with the site in its first week of operation [4].The negative publicity for President Barack Obama was so high, that the change of the health care system itself came into question. This is only one example that shows the importance of software systems in today's society and everyone's life.

As software systems are created, a development methodology is used and the system knowledge is documented in form of source code comments, design documentation, manuals, tutorials, test documents and other artifacts. One of the core challenges during creation and especially later during maintenance is to keep these artifacts up-to-date. Research has shown, that a lack of up-to-date documentation artifacts is one of the largest root. causes of defects in software systems [17]. It is also the primary reason for quick software system quality degradation and aging [9], [19].This paper intends to give a status of the current practices and their quality. The focus is on the technical documentation. The end user documentation, which describes the software system and its intended use, is not focus of this paper.

Stakeholders

Software system documentation has different stakeholders, which specific interests. Different authors [2; 5] have performed a stakeholder analysis on this topic. The following table provides a combined view on the stakeholders with their concerns.

The initial creation as a project mainly concerns:

- customer;
- developing Organization Management;
- architect;
- developer.

Nearly all development and project management methodologies e.g**.** Rational Unified Process (RUP) or Project Management Institute (PMI) addresses software system documentation well, an endless seeming number of artifacts are defined and their content gets revised multiple times during the initial system development. Where it is overall agreed that system documentation is very important, one of the reasons agile methods became so popular has been their focus on the software product itself. Working software is explicitly seen as more important than comprehensive documentation [1].Participants of software system development efforts became frustrated in the past of the amount of documents created in a project and their redundancy. Studies by Weskits [20] have shown a redundancy of nearly 50 % of content in documentation artifacts.

The ongoing operation and work with the software system and its maintenance concerns:

- user;
- maintainer.

User training and specific documentation in respect to the systems usage is a deliverable of the project and given to the users normally before the end of the project.

1. Stakeholder analysis based on studies [2; 5]

Stakeholder	Concerns
Customer	• Schedule estimation • Budgeting • Feasibility and risk management • Requirements traceability • Progress tracking
Developing Organization Management	• Schedule estimation • Budgeting (low costs, keeping people employed) • Feasibility and risk assessment • Requirements traceability • Progress tracking
User	• Consistency with requirements and usage scenarios • Non-functional requirements (performance, reliability etc.) • Accommodate future requirements
Architect	• Support of trade-off analysis • Requirements traceability • Completeness, consistency of architecture
Developer	• Sufficient detail for design • Reference for selecting/assembling components • Maintaining compatibility with existing systems
Maintainer	• Maintain compatibility with existing systems • Guidance on software modifications • Guidance on architecture evolution

Most development and project management methodologies address the early involvement of the maintainer as well. The reality in most companies however is that the maintainer is only involved in the late hand-over phase, if even. Explanations based on own experiences are organizational boundaries between development team and maintenance team and ignorance of the development teams about the importance of an early involvement of the maintainer. Studies by Kajko-Mattsson [8] have shown that the status of software system documentation already during the time of the handover between developments to maintenance is unsatisfactory.

Scope of system documentation

The IEEE defines software maintenance as the "process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment" [6]. Kajko-Mattsson presented requirements for software system maintenance documentation [8]:

- The system documentation is correct, complete and consistent, when the system is transferred from development to maintenance.
- The software system is documented at all granularity levels of system documentation during corrective maintenance.
- User manual is consistent with the state of the software system.
- Regression test case specification is revised and modified, if required.
- Traceability between all levels of system documentation is ensured.
- Traceability of change is ensured.
- Guidelines of documenting a software system are defined.
- Guidelines for internal documentation are defined.
- An organization has a checklist of all document types required for executing and supporting the corrective maintenance process.
- Corrective maintenance process explicitly states where in the process each type of documentation should be updated /checked.
- The corrective maintenance task is not approved as accomplished until the quality of all levels of system documentation for this task is inspected.
- The quality of the documentation is periodically checked by some documentation process manager.
- If not satisfactory or missing, the documentation of the existing software is updated as soon as it is possible.
- System development documentation and system development journal (experience documented) are available to the maintenance organization.
- All system changes are recorded during corrective maintenance.
- History of the reported software problem is recorded for each product/product component.
- System Maintenance Journal is kept for monitoring the corrective maintenance process.
- An organization arranges education/training in written proficiency for maintenance engineers.

As Kajko-Mattsson and earlier studies [17] have shown, those requirements are far away form being fulfilled. The main issue of software system documentation arises during ongoing operation and work with the software system. Studies by Secord [13] have shown that 90 % of the total cost of typical software systems accumulates during maintenance. A software system that is used will undergo changes or it will loose its usefulness over time. During those changes and regular maintenance tasks, the system documentation is besides the source code the only source of information for maintainers. Because of the lack of up-to-date information maintainers often have to work from the source code [8]. Studies [10; 11] have shown that 40 – 60 % of maintenance activities are spend on studying the software source code to understand how the planned modification may be implemented. A situation that companies are well aware of. This seems highly ineffective and has its reasons mainly in the state of the system documentation.

A study performed by Souza [14] asked maintainers, what artifacts they actually found important during their work.

2. Importance of documentation artifacts based on studies [14]

Artifact	Structured analysis	Object-oriented analysis
Source code	92 %	94 %
Comments	78 %	75 %
Logical data model	74 %	71 %
Physical data model	62 %	60 %
Class diagram	–	62 %
Requirements description	59 %	–
Use case diagram	–	58 %
Use case specification	–	50 %
Acceptance test plan	53 %	50 %
Requirements list	52 %	–
Data dictionary	46 %	46 %
Conceptual data model	44 %	43 %
Implementation plan	41 %	44 %
Unitary test plan	39 %	38 %
Data migration plan	35 %	38 %
Data flow diagram	35 %	–
System test plan	–	38 %
Sequence diagram	–	33 %
Functional prototype	32 %	26 %
Activity diagram	–	28 %
Component specification	29 %	16 %
Architectural model	28 %	–
Vision document	–	27 %
Context diagram	25 %	–
Hierarchical function diagram	23 %	–
Glossary	22 %	21 %
Non-functional prototype	19 %	22 %
Functions derived from requirements	22 %	–
State diagram	–	14 %
General transaction diagram	20 %	–

While the study shows some variation between structured analysis and object-oriented analysis, the overall picture and tendency is very similar.

The most important documentation artifact is the source codes itself and its documentation followed by the data model documentation. Surprisingly the study has shown, that architectural models and other generalized views of the software system, which are pushed by the literature as being highly important are not seen as important by maintainers. Other studies confirm those findings [16].

As it got shown that a lot of documentation is not up-to-date [9; 18] and therefore does not reflect the current state of the software system, this could be one explanation. Another reason could be trust, because it's hard for a maintainer to verify, that the documentation artifacts are actually up-to-date. So as the study has shown maintainers go back to the working source code and documentation elements that are tightly related to it, like the source code comments. The same is true for the data models, which can even be generated automatically out of the database system.

However there is a small difference between structured analysis and object-oriented analysis. As studies have shown [3] there is a slightly higher duration in using Unified Modeling Language (UML) for documentation for the purpose of software maintenance. Unified Modeling Language benefits slightly in terms of functional correctness (understanding and changes implemented on this base) and traceability of use cases to code.

Measurement

The maxim "What can't be measured, is hard to prove" by the quality guru Demin is also true in respect to software system documentation.

There is a variety of measurement approaches for software system documentation artifacts. Well in use are measurement approaches on a source code/comment level. Scheck [12] gives a good picture on those measures based on Java. Also in the industry those measures are used to verify source code documentation quality, code style/conventions and to assess the source code itself using source code metrics.

Much harder is the measurement of the quality of architectural models or other generalized views of the software system. Only the pure existence can be verified, their quality has to be assessed by a qualified reviewer, which involves time and effort. So based on own experience its seldom done in companies. But where is no control the overall quality degenerates.

Viscount [18] proposes with the Documentation Process Maturity Model (DPMM) a much broader concept following the idea of the Capability Maturity Model to measure quality. The model recognizes the challenge of tangible deliverables, however can't provide a solution. After working with 90 companies and evolutionary developing four versions of the model, Viscount concludes that the maturity level draw attention away from the key practices. Those key practices that Viscount identified for the fourth version are

- Creation of basic software documents.
- Management recognition of importance of documentation.
- Existence of documentation policy and standards.
- Monitor implementation of policy and standards.
- Existence of a defined process for creation of documents
- Methods to assure quality of documentation.
- Assessment of usability of documentation.
- Definition of software documentation quality and usability measures.
- Collection and analysis of documentation quality measures.
- Collection and analysis of documentation usability measures.
- Process improvement feedback loop.

During the usage of the model with 90 companies over five years, Viscount concludes with the fourth version that "…there is a considerable gap between the intentions and the actions…". The management is well aware of the importance of software system documentation and put policies and standards in place. However their usage is low, there is limited measurement, assessment and improvement.

Conclusion

Software system documentation is especially crucial during maintenance. This paper has shown, that current documentation practices are as Viscount puts it "…not getting a passing grade in the software industry…". The amount of documentation artifacts is too high, their content highly repetitive and even at the time of handover between developments to maintenance they are largely not accurate. Only a small portion of those artifacts is actually used during maintenance, mainly the source code itself, its comments as well as the data models. Those artifacts can also be measured well.

While the problem of poor software system documentation is known for some time, there seems no general answer and the research of the last decades has shown to be rather ineffective.

It might be time to re-discuss and go back to basics, looking at what maintainers actually use and are willing to keep up-to-date. This

seems a lot like the agile story for software development and project management, focusing on source code and the people who create the software system. We have seen that this became very effective also in reshaping established methodologies. What could be the learning's of the agile idea to documentation?

- Keep it active (alive).
- Keep it with the source code (automate).
- It's about the people (communication).

Maintainers have to be involved early enough during the development of the software systems. To break barriers between development team and maintainers (communication), we could also foster communication between developers and maintainers working or which have worked on the same software system. In addition we have seen that maintainers are only willing to work with and update a very limited number of artifacts. Those artifacts have to be close to the source code (as the single truth) (e.g. inline source code comments (automate)) and should wherever possible be able to kept current using automation (automate, alive). That source code related artifacts can be measured well, which lead to higher quality and trust. Maintainers are willing to use and would trust those artifacts rather than written static documents. The same is true for architectural models and other generalized views, if their creation is automated and their quality can be measured well.

We as researches have to focus on those facts to come up with standards and tools to provide better approaches and ensure that the result can be measured and assessed.

References

1. Beedle M., Bennekum A.V., Cockburn A., and others, (2001). The Agile Manifesto, http://http://agilemanifesto.org/ (accessed: 01.03.2014).

2. Dolan T, Weterings R., and Wortmann J.C. (1998), Stakeholders in Software-System Family Architectures, in: *Proceedings of the 2nd International Esprit ARES Workshop,* Las Palmas de Gran Canaria, Spain, (27.02.1998).

3. Dzidek J.W., Arisholm E., and Briand L.C. (2008). A realistic Empirical Evaluation of the cost and Benefits of UML in Software Maintenance, in: *IEEE Transactions on Software Engineering*, Vol. 34, No. 3.

4. Ford P. The Obamacare Website Didn't Have to Fail. How to Do Better Next Time. Bloomberg Businessweek, (accessed: 16.10.2013).

5. Gacek C., Abd-Allah A., Bradford C., and Boehm B. (1995), On the Definition of Software System Architecture, in: *ICSE 17th Software Architecture Workshop*, April 1995.

6. IEEE Standard 610.12:1990, Glossary of Software Engineering Terminology.

7. ISO/IEC/IEEE. Defining Architecture, http://www.iso-architecture.org/ieee-1471/defining-architecture (accessed: 28.02.2014).

8. Kajko-Mattsson M. (2005). A Survey of Documentation Practice within Corrective Maintenance, in: *Empirical Software Engineering*, 10, pp. 31– 55.

9. Parnas D.L.. (1994). Software aging, in: *Proceedings of the 16th International Conference of Software Engineering.* pp. 279 – 287.

10. Pfleegwe S.L. (2001), Software Engineering – Theory and Practice.

11. Pigoski T.M. (1996). Practical Software Maintenance – Best Practices for Software Investments.

12. Schreck D., Dallmeier V., Zimmermann T. (2007). How Documentation Evolves Over Time, in: *Proceedings of the ACM IWPSE 2007.*

13. Seacord R.C., Plakosh D., and Lewis G.A. (2003). Modernizing Legacy Systems – Software Ttechnologies, Engineering Processes and Business Practices.

14. Wojciech D.J., Arisholm E., and Briand L.C. A Realistic Empirical Evaluation of the costs and Benefits of UML in Software Maintenance.

15. Souza S.C.B., Anquetil N., and Oliveira K.M. (2005). A Study of the Documentation Essential to Software Maintenance, in: *Proceedings to ACM SIGDOC 2005.*

16. Stettina C.J., and Kroon E. (2013). Is there an agile Handover? Findings from an Empirical Study on Documentation and Handover Practices Across Agile Project Teams, in: *Proceeding of 2013 IEEE*

International Technology Management Conference & 19th ICE Conference 2013.

17. Tilley S.R., Müller H.A., and Orgun M.A. Documenting Software Systems with views, in: *Proceedings of the 10th annual International Conference on System Documentation (SIGDOC), Ottawa, ON*, Canada, 1992, pp. 211 – 219.

18. Visconti M., and Cook C.R. (1993). A Software System Documentation Process Maturity Approach to Software Quality, in: *Proceedings of the 11th Pacific Northwest Software Quality Conference, 1993*, pp. 257 – 271.

19. Visconti M., and Cook C.R.. (2002). An Overview of Industrial Software Documentation Practices, in *Proceedings of the 12th IEEE International Conference of Chilean Computer Science Society*, 179 – 186.

20. Wingkvist A., Ericson M., Lincke R., and Löwe W. (2010), A Metrics-based Approach to Technical Documentation Quality, in: *Proceedings of the 7th International Conference of Information and Communication Technology.*

Received 28.02.2014

Volker Herwig,
Prof. Dr.-Ing. of Faculty of Building Technologies and Applied Informatics at University of Applied Sciences Erfurt,
AltonaerStrasse 25, 99085 Erfurt,
E-mail:
Germanyvolker.herwig@
fh-erfurt.de