COMMUNICATION &LANGUAGE at work
Issue no. 3 | February 15th 2014
Social Media and/in Business Communication

Information Seeking & Documentation as Communication: A Software Engineering Perspective

Dr. Michael P. O'Brien

(pp. 26-37)

http://ojs.statsbiblioteket.dk/index.php/claw/article/view/16558

Information Seeking & Documentation as Communication: A Software Engineering Perspective

Dr. Michael P. O'Brien
School of Languages, Literature, Culture, and Communication
University of Limerick

Abstract

Effective communication of knowledge is paramount in every software organisation. Essentially, the role of documentation in a software engineering context is to communicate information and knowledge of the system it describes. Unfortunately, the current perception of documentation is that it is outdated, irrelevant and incomplete. Several studies to date have revealed that documentation is unfortunately often far from ideal. Problems tend to be diverse, ranging from incompleteness, to lack of clarity, to inaccuracy, obsolescence, difficulty of access, and lack of availability in local languages.

This paper begins with a discussion of information seeking as an appropriate perspective for studying software maintenance activities. To this end, it examines the importance and centrality of documentation in this process. It finally concludes with a discussion on how software documentation practices can be improved to ensure software engineers communicate more effectively via the wide variety of documents that their projects require.

Introduction

Much of the research carried out to date in the area of software maintenance practice, centres around 'software comprehension'. O'Brien (2003) defines software comprehension as *a process whereby a software engineer understands a software artifact using both knowledge of the domain and/or semantic and syntactic knowledge, to build a mental model of its relation to the situation*. Here, domain knowledge refers to the context of the problem that is addressed by a piece of software (Shaft, 1992).

Singer et al. (1997) demonstrate that software engineers spend a considerable amount of their time seeking information - in other words, at the core of any software

comprehension activity, is an information requirement. Over several studies, they found that searching for information is done far more often than any other activity (Singer et al., 1997), (Singer, 1998). The purpose of software documentation is to help software engineers with that (enormously expensive) task.

Essentially, documentation can be defined as any communicable material that is used to describe, explain or instruct regarding some attributes of an object, system or procedure, such as its parts, assembly, installation, maintenance and use. Typical examples of (technical) documentation include: user manuals, patents, design specifications, standards documentation, annual reports, etc. Documentation is especially important in the software engineering sector - and spans the entire software development lifecycle from requirements gathering to system maintenance and re-engineering.

When technical writers refer to software documentation, they usually tend to mean 'user manuals'. There is, however, much more to software (technical) documentation than simply just user manuals. *"Concepts are prepared and documented, requirements are derived and written into specifications, designs are created and recorded in design documents, test cases and results are documented in special test case documents, and finally the as-built product is described in maintenance manuals"* (Glass, 1989).

Acquiring accurate, up-to-date, and relevant, information about the system being maintained is an arduous task in itself, (Lethbridge & Singer, 2003) as oftentimes, the sources of information are limited, inaccessible, or even unknown (Kajko-Mattsson, 2001). Software engineers tend to rely on several different sources of information about the system they are attempting to correct, adapt, or perfect (O'Brien, 2008). These information sources range from requirements and design documentation, through the system itself (source code & execution) to user documentation (user manuals, etc.) and verbal/non-verbal communication with other users who have experience with the system under study (Seaman, 2002; Singer et al., 1997; Singer, 1998; Ko et al., 2007).

Along with these external representations, programmers also use their own accrued expertise (knowledge base) consisting of, for example, a general understanding of the domain(s) under study, company coding conventions, knowledge of the syntax and semantics of the programming language at hand. As their knowledge of the system evolves and deepens, the programmer's knowledge base is updated and thus expanded.

Information Search & Retrieval

Categories of Information

Sim (1998) refers to software practitioners as task-oriented information seekers,

focusing specifically on getting the answers they need to complete a task using a variety of information types. For example, when finding a bug or carrying out an enhancement to a software system, the programmer will perceive an information need (to locate the bug). The perceived need will lead them to search for information, making demands upon a variety of information sources. Jarvelin & Repo (1983, 1984), propose three categories of information software engineers use as they seek information relevant to their maintenance task(s), namely:

- **problem information**: information, which describes the structure, properties, and requirements of the problem at hand.

- **domain information**: which consists of known facts, concepts, laws, and theories in the domain of the problem. In the case of software maintenance, this could be considered to be the application domain addressed by the software system.

- **problem-solving information**: this type of information describes how problems should be seen and formulated, what problem and domain information should be used - and how it should be used, in order to solve the current problem.

These three information categories represent three different dimensions and have different roles in addressing a problem. All categories are necessary in problem treatment but, depending on the task, only some of the categories may be available to a software engineer.

Singer (1998) carried out an interview study at ten industrial sites to probe the work practices of software engineers (two engineers, who worked on the same system were interviewed from each company). Specifically she wished to assess/identify the information sources they found valuable. She found that:

- the source code is the primary source of information used by programmers when carrying out enhancements to software systems.

- when these respondents were asked about documentation as a communicative source, they stated that they only occasionally referred to it. The main reason given for the general distrust of documentation was the fact that documentation is *time consuming to create and maintain* and is therefore, often *incomplete*, and consequently, *untrustworthy*. Incidentally, the more abstract the documentation was, the more respondents trusted it. The reason for this is mainly due to the perception amongst software engineers that higher-level abstractions tend to remain useful when lower-level details become outdated.

Information Seeking

Problem solving can be defined as *thinking that is directed toward the solving of a specific problem that involves both the formation of responses and the selection among possible responses* (Solso, 1995). Information seeking is performed when forming these responses and selecting among possible responses.

The term information seeking often serves as an umbrella overarching a set of related concepts and issues (Kingrey, 2002). In the simplest terms, information seeking involves the search, retrieval, recognition, and application of meaningful content. This search may be based on specific strategies or serendipity, the resulting information may be embraced or rejected, and the entire experience may be carried through to a logical conclusion or aborted in midstream. Indeed, there may be a million other potential results.

Kingrey (2002) states that information seeking has been viewed as a cognitive exercise as a social and cultural exchange, as discrete strategies applied when confronting uncertainty, and as a basic condition of humanity in which all individuals exist. In fact, information behaviour may be a more encompassing term, rather than information seeking, to best describe the multi-faceted theory (Niedzwiedzka, 2003).

The process of information seeking is generally considered an individual activity. However, in many cases, people work in teams and have common information needs (for example, software developers). Poltrock et al. (2003) expand the definition of information seeking to include communicating about the information need, sharing the retrieved information within the team and coordinating the constituent information retrieval activities across multiple participants. Only in recent years have researchers begun looking at information retrieval as a collaborative activity (O'Brien, 2008). Like an individual, a team must recognise its information needs but must subsequently effectively communicate retrieved information to its members.

The information seeking process itself may lead to either a success or a failure. If the process is successful information can be located, which can be used in assisting in solving the problem. However, this may result in the satisfaction or non-satisfaction of the original perceived need. Satisfaction occurs when the located information has been analysed and placed in the context of the problem and fully satisfies the original need.

Non-satisfaction occurs when the information does not fully satisfy the original need. With non-satisfaction, information seeking may be iterated until satisfaction occurs (Hayden, 2000). A failure to find information may result in the process of information seeking being continued or refined. As Krikelas (1983) states, information seeking begins when someone perceives that the current state of knowledge is less than that

needed to deal with some issue (or problem). The process then ends when that perception no longer exists.

A typical example of an information seeking activity in a software maintenance context is presented in Table 1. As the programmer seeks-out and gathers all the relevant sources of information, she/he typically examines the results of their search in terms of usefulness in solving a given problem (in this case, a bug fix). The information seeking process is then complete when the programmer's information requirements are satisfied and she/he can make their modifications to the system (problem solution).

Table 1: Example Information Seeking Scenario

Awareness of Problem	*A bug report arrives on programmer's desk top*
↓	The programmer studies the bug report (which subsystem is it in, who was the user(s), what were they doing, or why did the system crash?)
↓	After some time, the programmer finds that the program crashed out at point X and then searches for the program called by that program before crashing
↓	She/he browses through the source code, seeks out the relevant system documentation and prioritises the available information in terms of its relevance to fixing the actual problem (bug)
↓	The programmer then searches for the chunk of code that contains the bug - and attempts to fix it
↓	She/he studies the recompiled executing code
Problem Solution	*The bug is now fixed and an incident report is generated*

Documentation as Communication

Effective communication means determining and providing answers to the complex problems of the real world (Albers, 2005). Although documentation plays a clear role in the communication process, its role and centrality is somewhat varied. For example, there are significant differences in documentation for a scientific study, computer software, patient records and legal cases.

Essentially, software documentation can be simply defined as *an artifact whose purpose is to communicate information about the software system to which it belongs* (Forward & Lethbridge, 2002). There are two broad categories of software documentation: 'technical' and 'user'. Technical documentation is primarily for software engineers who

develop or maintain computer systems; user documentation, on the other hand, is for those who use the computer system for a specific purpose.

In software engineering terms, documentation is much more than just textural descriptions of a software system. It is, in essence, a form of communication amongst team members - with communication only being achieved when information is ultimately being conveyed. Based on the assumption that documentation is communication, the goal of a particular software document is to convey information. Examples of typical communicative documents include:

- **Requirements documentation**: documents that identify attributes, capabilities, characteristics and qualities of a software system.

- **Design documentation**: overview of the software system (software architecture description).

- **Technical documentation**: documents detailing algorithms, code, interfaces, etc. Incidentally, it is these documents most software engineers refer to when using the umbrella term *software documentation*.

As previously stated, this information may not necessarily be completely accurate or consistent. Typical reasons for this include:

- the high cost of creating and maintaining good documentation
- the fact that programmers usually do not like to write documentation and often are not good at it
- the difficulty in finding people who are both skilled at writing and have a good understanding of the technology
- the vast amount of material that needs to be documented.

One of the most important forms of documentation for computer software is one that end-users never see - i.e. the comments that are included in the source code of computer programs. Source code is the version of software as it is originally written in a specific programming language (e.g. Java, C++). Comments do not affect the operation of the program in any way and are separated from the source code by special markers. They are statements by software developers explaining their code to other programmers who may work on the same programs and to remind themselves of what they did or what remains to be done. Comments also include justifications on why certain sections of code are written in a particular way and what they are intended to do.

The primary tool of software maintenance engineers is the body of software documentation. Software documentation helps keep the software running and up-to-date. Without clear and complete documentation of the software, the engineers must recreate the data on which they will base enhancement and correction actions.

Somerville (2010) states that document quality is equally as important as program quality. Without information on how to use a system or how to understand it, the utility of that system is degraded. Achieving document quality requires a definitive

commitment to document design, standards, and quality assurance processes. Producing good documentation is neither straightforward nor cheap and many software engineers find it significantly more challenging than producing good quality programs.

Importance of Good Documentation

The importance of documentation cannot be understated, especially when referring to the information search and retrieval processes of software maintenance engineers. In a software engineering environment, user requirements coming from the product management team need to be translated, dependencies among different entities have to be understood, models have to be prepared, justifications have to be given for design choices that are made, and architectures have to be conceived (Kumar, 2012). At the same time, software engineers need to answer questions like: how is the entire development team staying on the same page? How are we going to realise our goals in the fixed time limit? What about maintenance down the line? What about software testing? In essence, having access to good, accurate, and timely documentation, is the answer to all of the above questions.

Good documentation can serve several very important functions with regards to computer software. The most obvious being that it can make it easier to use and thereby save users' time, frustration and money. It can also be useful for the developers and retailers of software because it can reduce the need for time-consuming and expensive support. Furthermore, it can enhance the perceived quality of the product and thereby lead to increased sales and profit margins for commercial software. Kipyegen & Korir (2013) state that as documentation acts as direct evidence of all the procedures and activities involved in software development, individual documents need to be up-to date, complete, consistent and usable.

When it comes to the improving, extending and updating of software products, good documentation is paramount. Due to the fact that computer programs can be extremely complex, their original developers often forget what they were doing or thinking when they created them, particularly after the passage of long periods of time. Moreover, it is often the case that some or all of the original developers of a piece of software are no longer in practice, thus making it even more challenging to understand the original software in order to improve, extend, or update it. Several studies to date along with anecdotal evidence strongly suggest that the largest cost of software is not in its creation, but rather in maintaining it. *Good*, but more importantly, *accurate* documentation can ultimately help to reduce this overhead. It can also significantly assist programmers with reducing the amount time spent on information seeking activities.

Parnas (2011) identified seven benefits of good documentation; namely:

- easier reuse of old designs
- better communication about requirements
- more useful design reviews
- easier integration of separately written modules

- more effective code inspection
- more effective testing
- more efficient corrections and improvements.

In addition to the existence of good documentation, the actual process of creating the documentation itself can be advantageous if done concurrently during the coding/development process. This is because it can help uncover problems in the software at various stages of its development and therefore provide timely feedback to the developers for correcting and improving it. Documentation is needed in all phases of the development of a software product: from the requirements phase, through analysis, design, coding, testing, until delivery - and (future) maintenance - of the product (Aimar, 1998). For this reason, as depicted in Figure 1, documentation is an activity that needs to commence early in development and continue throughout the development lifecycle (Kipyegen & Korir, 2013).

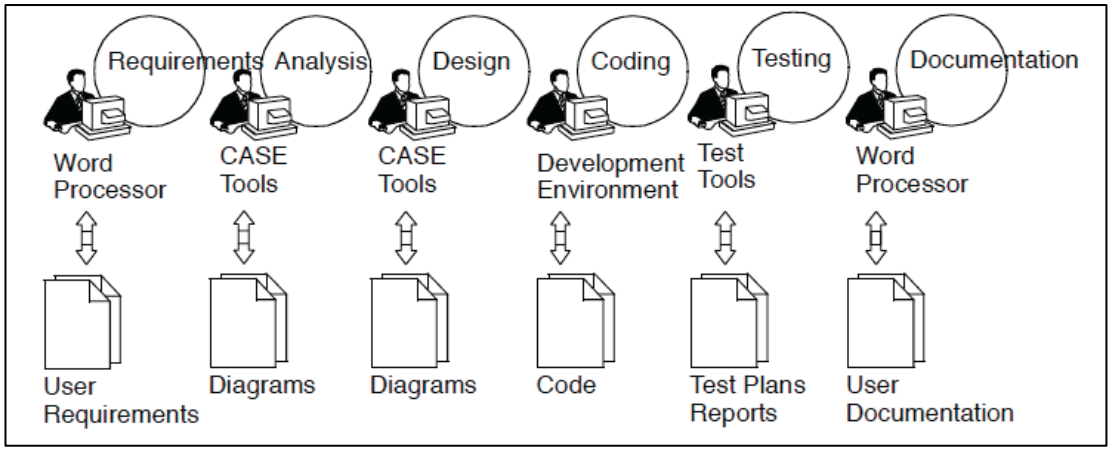

Figure 1: Documentation Produced During Software Development (Aimar, 1998)

It is crucial that the functional specifications of the system (specified by the client during the *user requirements* phase – see Figure 1) are documented accurately, signed-off, and carried through right up to the completion of a software development project. The quality of requirements documentation can have a significant impact on the outcome of any project. Any shortcomings here could potentially lead to a wide range of negative consequences, not only for the current project, but also for the business as a whole.

Conclusion

The prime purpose of documentation is communication. In a software engineering context, it provides information to people who are developing, maintaining or using computer systems. Understanding software systems is not limited solely to extracting

information from the source code alone. Any form of system documentation or previous maintenance records, are used as an integral part of the information search and retrieval process of software engineers. However, it is possible that this documentation will be inaccurate, inconsistent, incomplete or even non-existent. This leaves the maintenance engineer with the arduous task of attempting to recover not only the operation and structure of the software system, but many design issues and implicit assumptions made during its implementation. Recovering all this information from the source code itself is clearly not a trivial task.

Communication and documentation are key for the success of any software engineering project (Kipyegen & Korir, 2013). Good communication is paramount in this environment and occurs between the various team players on a daily basis by way of verbal, non-verbal and written communication.

Parnas (2011) boldly states that generally software engineers do not know how to produce *precise* documents for software. He further suggests that failure to document software designs properly reduces the efficiency of every phase in development lifecycle and contributes to the low quality software that we see so often today. When documents are produced, they generally tend to follow no defined standard and lack information that is crucial to make them understandable and usable by developers and maintainers (Briand, 2003).

In a consolidated effort to improve software documentation practices - from the earliest stage, one initiative would be to teach technical writing *within* computer science university programmes - and not in *separate* transferrable sills-type modules. If a separate writing skills module is offered to undergraduate computer science students (as a foundation), it should ideally be tailored to meet the writing objectives and needs of software engineers in practice; for example it could be taught in conjunction with a software development project where students are given a real-world type opportunity to write the kinds of documents they will ultimately write in industry.

Brockmann (1992) refers to *documentation as communication* designed to ease interactions between computer software and the individuals, who design, manage, operate, and/or maintain it. If emphasis is not placed on technical writing and the importance of producing good documentation at an early stage, it is unlikely that the graduate will prioritise the centrality of documentation as communication in an industrial context. With a clear foundation, understanding, and renewed appreciation of the writing process in general along with its associated cognitive components, software engineers will be better able to communicate effectively in the wide variety of documents that software development projects require.

References

Aimar, A. (1998). Introduction to Software Documentation. *Proceedings of CERN School of Computing*. Geneva, Switzerland.

Albers, M. J. (2005). The Key for Effective Documentation: Answer the User's Real Question. *Proceedings of the Annual Conference of the Society for Technical Communication*, 44, pp. 248-251.

Briand, L. C. (2003). Software Documentation: How Much is Enough? *Proceedings of the 7th European Conference on Software Maintenance and Reengineering, pp*. 13-15. IEEE Computer Society Press.

Brockmann, R. (1992). *Writing Better Computer User Documentation: From Paper to Online (2nd Edition)*. New York: John Wiley & Sons.

Forward, A., Lethbridge, T. (2002). The Relevance of Software Documentation, Tools and Technologies: A Survey. *Proceedings of the 2002 ACM Symposium on Document Engineering*, pp. 26-33.

Glass, R. (1989). Software Maintenance Documentation. *Proceedings of the 7th Annual Conference on Systems Documentation*. Pittsburg, PA, USA.

Hayden, K.A. (2000). Information Seeking Models. *EDCI 701*. University of Calgary. http://people.ucalgary.ca/~ahayden/seeking.html. Date accessed: November 2013.

Jarvelin, K., Repo, A., (1983). On the Impacts of Modern Information Technology on Information Needs & Seeking: A Framework, In H. J. Dietschmann, (Ed), *Representation and Exchange of Knowledge as a Basis of Information Processes* (pp. 207-230), Amsterdam, NL: North-Holland.

Jarvelin, K., Repo, A. (1984), A Taxonomy of Knowledge Work Support Tools. *Proceedings of the Annual Meeting of the American Society for Information Science*, Vol. 21, pp. 59-62.

Kajko-Mattsson, M. (2001). The State of Documentation Practice within Corrective Maintenance. *Proceedings of the IEEE International Conference on Software Maintenance*, pp. 354-363.

Kingrey, K. P. (2002). Concepts of Information Seeking and Their Presence in the Practical Library Literature. *Library Philosophy & Practice*, Vol. 4, No. 2.

Kipyegen, N., Kori, W. (2013). Importance of Software Documentation. *International Journal of Computer Science Issues*, Vol. 10, Issue 5, No. 1, pp. 223-228.

Ko, A. J., DeLine, R., Venolia, G. (2007). Information Needs in Collocated Software Development Teams. *Proceedings of the International Conference on Software Engineering (ICSE)*. Minneapolis, USA.

Kumar, V. (2012). *The Importance of Software Documentation on IT Product Companies*. Available online: http://askvinay.com. Date accessed: November 2013.

Lethbridge, T., Singer, J. (2003), How Software Engineers use Documentation: The State of the Practice, *IEEE Software*, Vol. 20, No. 6, pp. 35-39.

Niezwiedzka, B. (2003). A Proposed General Model of Information Behavior. *Information Research*, Vol. 9, No. 1.

O'Brien, M. P. (2003). Software Comprehension – A Review & Research Direction. *Technical Report UL-CSIS-03-3*. University of Limerick, Ireland.

O'Brien, M. P. (2008). Empirically Evolving a Model of the Information-Seeking Behaviour of Industrial Programmers. *Unpublished Ph.D. Thesis*. University of Limerick.

Parnas, D. (2011). Precise Documentation: The Key to Better Software. In Nanz, S. (ed.) *The Future of Software Engineering*, pp. 125–148. Springer, Heidelberg.

Poltrock, S., Gudrin, J., Dumais, S., Fidel, R., Bruce, H., Pejtersen, A. (2003). Information-Seeking & Sharing in Design Teams. *Proceedings of the International Conference GROUP '03*, pp. 239-247.

Seaman, C. (2002). The Information Gathering Strategies of Software Maintainers. *Proceedings of the International Conference on Software Maintenance*. pp. 141-149.

Shaft, T. M. (1992). The Role of Application Domain Knowledge in Computer Program Comprehension and Enhancement. *Unpublished Ph.D. Thesis*. Pennsylvania State University.

Sim, S. E. (1998). Supporting Multiple Program Comprehension Strategies During Software Maintenance. *Unpublished MSc Thesis*. Department of Computer Science, University of Toronto.

Singer, J., Lethbridge, T., Vinson, N., Anquetil, N. (1997). An Examination of Software Engineering Work Practices. *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative research*. Toronto, Canada.

Singer, J. (1998). Work Practices of Software Maintenance Engineers. *Proceedings of the International Conference on Software Maintenance*. Washington, Federal District of Columbia, USA. pp. 139-145.

Solso, R. L. (1995). Cognitive Psychology. Boston, MA: Allyn and Bacon
Sommerville, I., (2004). *Software Engineering (7th Edition)*. Addison-Wesley.

Sommerville, I. (2010). *Software Engineering (9th Edition)*. Addison-Wesley.

Author

Dr. Michael P. O'Brien

Teaching Assistant, School of Languages, Literature, Culture, and Communication, University of Limerick, Ireland.

Dr. Michael P. O'Brien holds a PhD in Computer Science from the University of Limerick and holds both an honours BSc degree in Information Systems and an MSc degree in Computer Science (by research & thesis). His research interests include cognitive and educational psychology, software comprehension strategies, empirical studies of programmers and software evolution. His award-winning research has been published at international conferences, workshops & seminars in this domain. Michael is a Member of the Irish Learning Technology Association, the Psychology of Programming Interest Group and the Computer Science Education Research Network. He is currently employed as an academic by the University of Limerick and lectures in the general areas of technical communication and instructional design.

Contact:

michaelp.obrien@ul.ie

School of Languages, Literature, Culture, and Communication,
University of Limerick, Castletroy, Co. Limerick, Ireland.