The Relevance of Software Documentation, Tools and Technologies: A Survey

Andrew Forward
University of Ottawa
800 King Edward
Ottawa, Ontario, Canada K1N 6N5
+ 1 613 255 3492
aforward@site.uottawa.ca

Timothy C. Lethbridge
University of Ottawa
800 King Edward
Ottawa, Ontario, Canada K1N 6N5
+ 1 613 562 5800 x 6685
tcl@site.uottawa.ca

ABSTRACT

This paper highlights the results of a survey of software professionals. The survey was conducted in the spring of 2002. The results are compiled from 48 individuals in the software field ranging from junior developers to managers and project leaders. One of the goals of this survey was to uncover the perceived relevance (or lack thereof) of software documentation, and the tools and technologies used to maintain, verify and validate such documents. The survey results highlight the preferences for and aversions against software documentation tools. Participants agree that documentation tools should seek to better extract knowledge from core resources. These resources include the system's source code, test code and changes to both. Resulting technologies could then help reduce the effort required for documentation maintenance, something that is shown to rarely occur. The data reports compelling evidence that software professionals value technologies that improve automation of the documentation process, as well as facilitating its maintenance.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance, and Enhancement**]: *Documentation*

General Terms

Documentation, Experimentation, Human Factors, Measurement, Performance

Keywords

Software documentation, documentation technologies, software engineering, software maintenance, program comprehension, documentation relevance.

1. INTRODUCTION

This paper presents the results of a survey of professionals in the software industry. The survey was conducted in April and May of 2002. This survey was constructed to uncover:

- The current industrial application of software documentation
- The industrial likes and dislikes of current documentation related technologies as well as the opinions regarding potential ones.

By software documentation, we are referring to any artifact whose purpose is to communicate information about the software system to which it belongs.

Common examples of such documentation include requirement, specification, architectural, and detailed design documents. These documents are geared to individuals involved in the production of that software. Such individuals include managers, project leaders, developers and customers.

Documentation attributes describe information about a document beyond the content provided within. Example attributes include the document's writing style, grammar, extent to which it is up to date, type, format, visibility, etc. Documentation artifacts consist of whole documents, or elements within a document such as tables, examples, diagrams, etc. An artifact is an entity that communicates information about the software system.

1.1 Motivation

During our interactions with software professionals and managers, it was observed that some large-scale software projects had an abundance of documentation. Unfortunately, little was understood about the organization, maintenance and relevance of these documents.

A second observation was that several small to medium-scale software projects had little to no software documentation. Individuals in these groups said they believed in the importance of

viduals in these groups said they believed in the importance of documentation, but timing and other constraints left few resources to document their work.

The primary questions arising from the above interactions are:

- How is software documentation used in a project?
- How does that set of documents favorably contribute to the software project (such as improving program comprehension)?
- How can technology improve the use and usefulness of such documentation?

One of the large-scale projects was seeking practical solutions to organizing and maintaining document information. Meanwhile, the smaller projects were looking for the benefits of documentation from both a value-added and a maintenance perspective.

In search of answers, a systematic survey was performed to question the thoughts of software practitioners and managers. Our approach is to build theories based on empirical data as opposed to mere intuition and common sense.

1.2 History

This paper covers the second survey on this topic. The first survey was conducted in the winter of 2002 and served as a pilot study. The winter survey participants were sampled from a fourth year software engineering course offered at the University of Ottawa. Although most participants did have some experience in the software industry, this survey was used primarily as a feedback mechanism for the questions themselves.

The April survey featured fewer and more concise questions with an improved sampling approach. All participants had at least one year of experience in the software industry; several had over ten years experience.

A summary of the data used in this report is available on-line [5]. Individual responses and identifying information have been withheld to protect confidentiality. The University of Ottawa's Human Subjects Research Ethics Committee approved the conducting of the survey.

1.3 Importance

The survey results presented in this paper are important for various reasons and to several audiences:

- Individuals interested in documentation technologies can use the data to better understand which existing technologies are useful and which are not, and why.
- This same information can be used to design tools and support features that improve the documentation process.
- Software decision makers can use the data to justify the use and selection of certain documentation technologies to best serve the information needs of the team.

1.4 Outline

The remainder of this paper is organized as follows:

- Section 2 describes the method under which the survey was conducted and the way in which we categorized participants based on their responses.
- Section 3 highlights several interesting findings from the gathered data.
- Section 4 summarizes the participants' demographics based on professional experience in the software industry.

2. SURVEY METHOD

2.1 Question Topics

The survey consisted of 48 questions of various types including multiple-choice, short answer, ratings, and free-form questions.

The question topics included …

- The role of software team members in the process of writing, maintaining and verifying different types of documentation.
- The participant's personal use and preference for different types of documentation, as well as opinions concerning the effectiveness of these.
- The ability of a document's attributes, as opposed to its content, to promote (or hinder) effective communication.
- The state of software documentation in the participant's organization.
- Comparison of past projects to current ones.
- The effectiveness of documentation tools and technologies.
- Demographics of the participants.

2.2 Participants

Participants were solicited in three main ways. The members of the research team approached:

- Management and human resource individuals of several high-tech companies. They were asked to approach employees and colleagues to participate.
- Peers in the software industry.
- Members of software e-mail lists. They were sent a generic invitation to participate in the survey.

Most participants completed the survey using the Internet [5]. A few replied directly via email.

There was a total of 80 responses to the survey. Of these, 48 surveys were complete and contained valid data.

The participants were categorized in several ways based on software process, employment duties and development process as outlined below.

We divided the participants into two groups based on the individual's software process as follows:

- Agile. Individuals that somewhat (4) to strongly (5) agree that they practice (or are trying to practice) agile software development techniques, according to question 29 of the survey.
- Conventional. Individuals that somewhat (2) to strongly (1) disagree that they practice agile techniques, or indicated that they did not know about the techniques by marking 'n/a' for not applicable.

In addition, we divided the participants based on current employment duties as follows:

- Manager. Individuals that selected manager as one of their current job functions, according to question 44 of the survey.
- Developers. Individuals that are non-managers and selected either senior or junior developer as one of their current job functions, according to question 44 of the survey.

Finally, we divided the participants based on management's recommended development process as follows:

- Waterfall. Individuals in the waterfall group selected waterfall as the recommended development process, according to question 46.
- Iterative. Individuals in the iterative group are non-waterfall participants who selected either iterative or incremental as the recommended development process, according to question 46.

3. SURVEY RESULTS

3.1 Technologies in Practice

This section highlights several documentation tools with which the participants have had experience. The participants listed technologies they found useful, and ones not so useful.

Question 36 asked,

> *Which software tools do you find MOST helpful to create / edit / browse / generate software documentation? (For example, text editors, word processors, spreadsheets, JavaDoc).*

Table 1 outlines the most frequently cited technologies based on 41 participants that answered question 36.

Table 1: Useful Documentation Technologies

Documentation Technology	Frequency	Percentage of Participants
MS Word (and other word processors)	22	54
Javadoc and similar tools (Doxygen, Doc++)	21	51
Text Editors	9	22
Rational Rose	5	12
Together (Control Centre, IDE)	3	7

Other technologies that participants found useful include ArgoUML, Visio, FrameMaker, Author-IT, whiteboards and digital cameras, JUnit and XML editors.

Question 37 asked,

> *Which software tools do you find LEAST helpful to create / edit / browse / generate software documentation?*

Several of the tools listed as most helpful by many participants were selected as least helpful by a few. These tools included MS Word and word processors (15% said they were least useful), JavaDoc and similar tools (12%), text editors (7%) and Rational Rose (2%).

In general, two types of technologies emerged as the most helpful for software documentation:

- Word and text processors. Although perhaps not the most efficient means of communication, these processors are flexible and in general easy to use.
- Automated documentation tools. These tools improve document maintenance by practically removing the need for it.

An interesting comment from one of the participants about documentation technologies (extracted from question 37) was:

> "The purpose of documentation is communication. Some tools are overapplied and the communication factor is lost. For example, a low level design tool should be easy to use in a brainstorming type of scenario when developers are hashing out the way to do something (currently, whiteboards are very effective for such interactions). If a low level design tool thinks [sic] its artifacts are an essential part of the software documentation to be maintained rigorously beyond that collaboration session, then that tool has an unwelcome fault."

This individual believes strongly in the purpose of documentation being that of communication. As such, some documentation efforts have a finite lifetime. The individual describes a low-level design tool as such an example. The participant believes that low-level design tools which over-emphasize maintenance are severely faulted. The ideas of document lifetime and over-emphasis on maintenance will be further explored in future sections of this paper.

3.2 Documentation Maintenance?

This section illustrates the extent to which documentation is maintained. The data presented below substantiates the claim that software documentation is rarely, if ever, updated. This information will serve as the basis for several other sections in this paper.

Question 4 asked,

> *In your experience, when changes are made to a software system, how long does it take for the supporting documentation to be updated to reflect such changes?*

The participants answered this question for the following types of documents

Requirements, Specifications, Detailed Design, Low Level Design, Architectural, Testing / Quality Documents

The participants selected from fixed values ranging between '*updates are never made'* (score of 1) and '*updates are made within a few days of the changes'* (score of 5).

Table 2 illustrates the preferred (mode) score, the percentage of responses of that score as well as the textual meaning of the score.

Table 2: How often is documentation updated?

Document Type	Mode	% of Mode	In Words
Requirements	2	52	Rarely
Specifications	2	46	Rarely
Detailed Design	2	42	Rarely
Low Level Design	2	50	Rarely
Architectural	2	40	Rarely
Testing / Quality Documents	5	41	Within days

Question 20 also pertained to documentation maintenance and asked to what degree to the participant agreed with:

Documentation is always outdated relative to the current state of a software system.

Many participants somewhat agreed (43%) with that statement, and a considerable number strongly agreed (25%).

The fact that documentation is infrequently updated does not imply that our sample participants work on projects of lower quality or that proper software engineering practices are not in place. In fact, very low correlation may exist between documentation maintenance and project quality [5].

The evidence that documentation is rarely updated is extremely important from a technology perspective. Since usage of tools that support documentation maintenance will be sporadic at best, such tools must enable users unfamiliar with a document to quickly comprehend its structure and content so they can make consistent and correct changes <1>. The tools must also be efficient from a task perspective, helping users to quickly accomplish what they intended to achieve <2>.

3.3 Evolving Documentation Needs

This section affirms the fact that the information needs of software professionals evolve over the lifetime of a project [1], [4].

Question 27 asked to what extent participants agreed that

Software documentation that I have found useful during inception / construction of a system differs from that which I find useful during maintenance and testing of that system.

The participants rated the question as follows: strongly disagree (1), somewhat disagree (2), indifferent (3), somewhat agree (4), and strongly agree (5).

Overall, many participants strongly (32%) and somewhat (46%) agreed that their needs for different types of documentation change throughout a project's lifecycle. Only a small portion of participants strongly disagreed (7%) with the statement. The results are consistent among all participant categories outlined in Section 2.2.

Question 22 asked to what degree the participant agreed that

Most software documents have a finite useful lifetime and should be subsequently discarded (or removed from the primary document repository).

Table 3 compares the percentage of individuals that disagreed (score of 1 or 2) with those that agreed (score of 4 or 5) with question 22 based on the categories outlined in Section 2.2.

Table 3: Does documentation have a finite lifetime?

Participant Category	% Disagree	% Agree	Sample Size
All	50	45	46
Agile	52	44	25
Conventional	59	41	17
Manager	50	42	12
Developer	47	47	17
Waterfall	54	46	13
Iterative	46	54	13

It is important to note that few participants had neutral opinions about this question and in general there was a strong split of opinion. Another interesting point of analysis is that question 22 contains two sub-questions asking

- Do documents have a finite useful lifetime?
- If so, should they be discarded afterwards?

As mentioned, from question 27 we see that the document needs of most participants changes over time. This suggests that these individuals would also agree that a document has a finite useful lifetime. If this is true, then the disagreement in question 22 most likely stems from how these documents should be treated once they are no longer used, either archived or discarded. The fear to discard documents may be the result of not knowing if, how, or when others might use a particular document. There may also be a reluctance to discard documentation due to the effort and resources required to produce it. It has been shown that archiving all documents and related artifacts has proved to be unsuccessful [7] – it tends to be impossible to search and use such a collection.

If few resources were expended to produce a document, then perhaps individuals would be less hesitant to discard such documentation. As well, if archiving documentation could be based on

usage, and its management somewhat automated, then improved navigation would be possible, improving efforts to retrieve pertinent and hence better documentation [10]. In later sections, we will provide additional insight into a document's useful lifetime.

3.4 Documentation Usage

This section highlights which types of documents are most used and by whom.

Question 6 asked,

> *In your experience, how often do you consult the available software documentation when working on that software system? Rate between one (1) as NEVER and five (5) as ALWAYS.*

In general and as expected, the results were diverse varying from never to always. Overall, the most popular document was the specification document, whereas quality and low-level documents were the least consulted (mean of 2.96, st. dev 1.31).

Table 4 lists the most used documents based on the categories outlined in Section 2.2.

Table 4: Most Used Documents

Participant Category	Mean	St Dev	Most Used Document Type
All	3.85	1.29	Specifications
Waterfall	3.88	1.13	Testing / QA
Iterative	4.50	1.00	Specifications
Agile	3.47	1.30	Specifications
Conventional	4.38	1.19	Specifications
Manager*	3.60	1.67	Requirements
Developer*	4.33	1.12	Architectural

* Statistically significant difference between means of a pair of participant categories with 95% confidence

Most categories referenced specification documents most often, even though these documents are rarely updated as shown in Section 3.1.

Although one would not argue that up-to-date documents are preferred, is it a requirement for useful and relevant documentation?

3.5 Support for document automation

This section describes two viable approaches to improve documentation maintenance technology.

Question 24 asked to what extent you agree that

> *Software documentation contains a lot of information that can be extracted directly from the system's source code itself.*

Question 30 asked to what extent you agree that

> *Automated testing (such as J-Unit) helps exhibit the true state of a system and is a useful tool for software documentation.*

Overall, many participants strongly (22%) or somewhat (37%) agreed with the statement that a lot of information can be extracted directly from the source code. As well, 23% strongly and 49% somewhat agreed that automated testing provides resources that serve as useful documentation. Few participants (11% and 5% respectively) were strongly against the ideas above.

The evidence suggesting that test code contains a lot of useful data has important implications including:

- First, it may be useful for information extraction tools to consider analyzing test source code for the purpose of documentation.
- Second, documenting results of automated system testing may better communicate the true features of a system.

Participants generally support the idea of improving document automation, and more research is required to determine what data should be extracted, and by what means.

3.6 Need for document tracking

Although more information should be extracted from the source code (as shown above), the process cannot be entirely automated. As one participant described "they [automated documentation tools] don't collect the right information."

This section describes the concept of tracking changes in a software project for the purpose of documentation maintenance. For instance, as changes are made to a system's source code, then all relevant documentation that refers to that code would be *marked* as potentially requiring updating.

Question 31 asked to what extent you agree that

> *It would be useful to have tools to track changes in a software system for the purpose of updating and maintaining its supporting documentation.*

An overwhelming number of participants strongly (42%) and somewhat (40%) agreed with that statement, whereas only a relative few (7%) disagreed.

Based on the data from this survey, the following high-level core requirements should be fulfilled in any technology relating to the above tracking mechanism.

First, the technology should complement existing documentation tools as well as attempt to seamlessly integrate itself into a project. From Section 3.1, we see that a variety of tools are employed with respect to documentation and it is unlikely that individuals will adopt new technologies if it means they must abandon what they currently use. As well, this technology should not focus on documentation, but rather to track changes between documents and source code. Finally, all team members should

have access to, as well as the ability to influence the information gathered by the technology.

Second, the technology must be able to relate documents to source code, as well as to other documents. When changes occur to source code, we cannot predict where changes may be required in the documentation unless appropriate relationships exist between the two. To support such relationships, documentation technologies should learn from the concept of *authoritativeness* in hyperlinked environments [10]. This concept suggests that documents containing the highest number of outgoing references, or are referenced the most, are the ones most likely to be worthy of attention. In addition, it is important that users be able to easily input their own relationships among documents, as well as to remove any automatically generated ones that appear irrelevant. This manual intervention helps to improve the quality of the relationships between documents and source code.

Third, the technology must be able to recommend possible discrepancies between relationships once changes in source code and other documents occur. The intent of this technology is to track changes for the purpose of maintenance. As such, a key requirement is the ability to recommend where changes may be required. This process should also support user intervention for the same reason as cited above.

These requirements should help improve document quality and maintenance for the following two important, yet distinctive reasons:

- First, possible inconsistencies between documents and source code are highlighted for the user, helping to provide a maintenance map when updating documentation.
- Second, maintenance can be prioritized based on user feedback as opposed to by assumption. Discrepancies will be highlighted in two main ways: automatically based on changes to the software or manually through user intervention. As complete document maintenance is impractical, maintainers can first concentrate on user feedback. Based on the notion that out-dated documentation can still be a useful[5], one can then base maintenance on user feedback regarding document inconsistencies as opposed to solely on the fact that a document may be outdated.

3.7 Supporting Lightweight Documents

This section brings together several key pieces of information to present a foundation in support of lightweight [1], everyday documentation.

Key features of lightweight technologies include supporting

- Content creation over maintenance. Documents are rarely maintained. As such, technologies should be easy to use and support the creation of information as opposed to its maintenance.
- Ideas over accuracy. It has been shown [5] that documentation best serves as a communication medium as opposed to a fact base that must be precise and up-to-date. As such, tools should facilitate the communication of ideas, sharing with others, and prompt feedback. For example, the use of whiteboards in combination with digital cameras or printing capabilities have been known to be an effective means to create documentation [4].
- Simplified features. The tools most preferred for documentation include word processors and text editors (see Section 3.1). These technologies provide extensive freedom to users and should be a role model for most documentation tools.
- Automated archiving. As seen in Section 3.3, many individuals are unlikely to discard documentation. However, too much documentation can decrease the usefulness of the entire repository [7]. As such, it would be beneficial to have lightweight technologies archive such documents based on user preferences and document usage.
- Reader feedback. Feedback is an important tool of communication [1],[4]. As a consequence, the easier the reader can provide feedback, the more attuned the writer can become at providing useful content.

3.8 Project Size Independence

This section provides evidence that the conclusions drawn in previous sections are independent from the project size (based in thousands of lines of code, KLOCs).

Question 41 asked,

> *What is the size of your current (or recently completed) project in KLOCS.*

The participant then selected one answer from the following list:

> *< 1 KLOC (KLOC = 1000 lines of code),*
> *between 1 and 5 KLOCS,*
> *between 5 – 20 KLOCS,*
> *between 20 – 50 KLOCS,*
> *between 50 – 100 KLOCS,*
> *over 100 KLOCS,*
> *or N/A.*

Table 5 illustrates the project size distribution for all categories outlined in Section 2.2.

Table 5: Project Size Distribution in Thousands of Lines of Code (KLOCs)

Participant Category	% of projects between 1 and 20 KLOCS	% of projects >= 50 KLOCS	Number of Individuals considered
All	29	35	45
Waterfall	36	44	13
Iterative	31	39	13
Agile	36	44	25
Conventional	24	18	16
Manager	33	50	12
Developer	35	35	17

As we see above, all categories were well represented. It is interesting to point out that a larger then expected portion of agile participants are working on large projects (agile development is typically associated with small projects). In fact, 32% stated they are currently working on projects over 100 KLOCs. Our phrasing of practicing agile techniques helps explain this high percentage. In the context of our research, individuals were asked if they practice agile techniques. Agreement with this statement does not necessarily imply that the project itself is agile. As such, it is not unfounded to have such a large portion of agile techniques applied to large projects.

The data from question 4 suggests low correlation between the project size and the participant categories as outlined in Section 2.2. As such, the results in other sections should hold regardless of project size.

4. DEMOGRAPHICS

In this section, we will describe the participants' demographics. The divisions separate individuals based on software experience, current project size and software duties. The purpose of this section is to show that the survey was broad-based, and therefore more likely to be valid.

Table 6 illustrates the participant's experience in the software field (based on number of years in the industry).

Table 6: Participants' Software Experience

Software Experience (years)	Number of Participants	Percentage
< 1	0	0
1 to 4	11	23
5 to 10	14	30
> 10	22	47

Table 7 indicates the current job functions held by the participants. Please note that one individual can have several functions.

Table 7: Participants' Employment in the Software Field*

Job Functions	Number of Participants	Percentage
Sr. Software Developer	19	40
Software Architects.	17	36
Project Leader	14	30
Manager	12	26
Technical Writers	10	21
Quality Assurance	9	19
Jr. Software Developers	5	11
Other	4	9
Software Support	3	6
None of the above	3	6
Student	1	2

* Note that many participants performed one or more function.

It appears from the above data that most employment areas in the software field have been well represented. The two somewhat under-represented categories are Junior Developers and Software Support. This survey was not directed at students since they probably would have lacked the experience to provide useful results.

5. SUMMARY

The data from the April 2002 survey of software professionals provides concrete evidence that debunk some common documentation misconceptions and lead to the following conclusions.

- Document content can be relevant, even if it is not up to date. (However, keeping it up to date is still a good objective). As such, technologies should strive for easy to use, lightweight and disposable solutions for documentation.
- Documentation is an important tool for communication as opposed to simply a fact sheet about the source code that is only relevant if well maintained. As such, technologies should enable the user to quickly and efficiently communicate ideas as opposed to providing strict validation and verification rules for building facts.

The conclusions given above, as well as the data presented in the paper, will help documentation technologies better serve the needs of writers and readers. As well, decision makers will be able to choose more appropriate documentation strategies and technologies based on needs as opposed to generic expectations.

Once we can admit that documentation is out-dated and inconsistent, we can then appreciate and utilize it as a tool of communication. This tool can then be judged based on its ability to communicate as opposed to merely presenting facts.

Software documentation should focus more on conveying meaningful and useful knowledge than on precise and accurate information.

5.1 Future Work

Based on these findings as well as the additional questions raised from this survey, the list below provides some possible avenues for continued research in this field.

- Technologies to track changes in a software system for the purpose of documentation maintenance.
- Technologies to rank and recommend documentation based on factors beyond readability, such as authoritative techniques.
- Technologies to extract documentation from test code.

As we learn more about how technology can improve documentation, we hope for improved techniques to communicate information about a software system in a clear and concise manner. Research in this area could widen our definition of documentation beyond just documents.

ACKNOWLEDGMENTS

Our thanks to all participants and participating companies (who must remain anonymous). Thank you to members of the Knowledge Based Reverse Engineering (KBRE) group at the University of Ottawa. Your feedback and support have been greatly appreciated.

A sincere thank you to Jayne Forward for helping edit this paper.

REFERENCES

[1] Ambler, Scott and Ron Jeffries. *Agile Modelling: Effective Practices for Extreme Programming and the Unified Process*, John Wiley & Sons, 2002, chapter 14.

[2] Angerstien, Paula. *Better quality through better indexing*, SIGDOC '85, Cornell University, Ithaca, New York, USA, ACM Press, p 57.

[3] Berglund E. (2000) Writing for Adaptable Documentation, IPCC/SIGDOC 2000, September 24-27, Cambridge, Massachusetts, p497 – 508.

[4] Cockburn, A. *Agile Software Development*, Addison-Wesley Pub Co, 2001.

[5] Forward, A. Software Engineering Documentation Priorities: An Industrial Study, submitted to CASCON 2002, available from [6].

[6] Forward, A. Survey data website available at www.site.uottawa.ca/~aforward/docsurvey/

[7] Glass, R. *Software maintenance documentation*, SIGDOC '89, Pittsburg, Pennsylvania, USA, ACM Press, p18 – 23.

[8] Jazzar, Abdulaziz and Walt Scacchi. *Understanding the requirements for information system documentation: an empirical investigation*, COOCS `95, Sheraton Silicon Valley, California, USA, ACM Press, p268 – 279.

[9] Klare, George R. *Readable computer documentation*, ACM JCD, Volume 24, Issue 3 (August 2000), p148 – 167.

[10] Kleinberg, J. *Authoritative sources in a hyperlinked environment*. Proc. 9th ACM-SIAM Symposium on Discrete Algorithms, 1998.

[11] Medina, Enrique Arce. *Some aspects of software documentation*, SIGDOC '84, Mexico City, Mexico, p57 – 59.

[12] Ouchi, Miheko L. Software Maintenance Documentation, SIGDOC'85, Ithaca, New York, USA, ACM Press, p18 – 23.

[13] Redish, Janice. *Readability formulas have even more limitations than Klare discusses*, ACM JCD, Volume 24, Issue 3 (August 2000), p132 – 137.

[14] Scheff, Benson H. and Tom Georgon. *Letting software engineers do software engineering or freeing software engineers from the shackles of documentation*. SIGDOC '88, Ann Arbor, Michigan, USA, ACM Press, p81 – 91.

[15] Stimely, Gwen L. *A stepwise approach to developing software documentation*, SIGDOC '90, Little Rock, Arkansas, USA, ACM Press, p122 – 124.

[16] Thomas, Bill and Scott Tilley. *Documentation for software engineers: what is needed to aid system understanding?*, SIGODC '01, Sante Fe, New Mexico, USA, p 235 – 236.