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. 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. Our 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

Keywords

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

DocEng'02, November 8-9, 2002, McLean, Virginia, USA.

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, among other things:

- The current industrial application of documentation in software projects.
- The likes and dislikes of software practitioners regarding documentation-related technologies as well as their opinions about potential improvements.

The documentation we consider in this research includes any artifact whose purpose is to communicate information about the software system to which it belongs, to individuals involved in the production of that software. Such individuals include managers, project leaders, developers and customers.

The forms of documentation we consider therefore include requirements, specifications, architectural documents, detailed design documents, as well as low level design information such as source code comments.

In this paper we will discuss various *documentation attributes*. These 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* are entities that communicate information about the software system; they include whole documents or elements within a document such as tables, examples, diagrams, etc.

1.1 Motivation

During our interactions with software professionals and managers, we first observed that some large-scale software projects had an abundance of documentation. These individuals agreed that 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 documentation, but timing, budget and scheduling constraints left few resources for these individuals to adequately 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 sought 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, we performed a systematic survey to question the thoughts of software practitioners and managers. Our approach is to build theories based on empirical data; possibly uncovering evidence that questions our intuition and common sense about documentation and its role in software engineering.

1.2 Related Work

Curtis et al [5] interviewed personnel from 17 large software projects. Their analysis was focused on the problems of designing large software systems, but many results report directly about the use (and mis-use) of documentation in a software project.

Our work provides statistical data that affirm some of the documentation issues Curtis identified.

Abdulaziz Jazzar [10] conducted an empirical investigation using a comparative case study research method. The basis for the work was concerned with the requirements for information system documentation.

Jazzar's work resulted in eight hypotheses that attempt to model the requirements for achieving effective, high quality documentation products and processes.

Our work complements Jazzar's as we focused on the attributes of quality documents, whereas Jazzar focused on the process of quality documentation.

In addition, our work contributes knowledge about several other facets of documentation including the current use and perception of software documentation tools and technologies.

1.3 Background

Before conducting the main survey described in this paper, we conducted a pilot study to help develop and refine the questions. The pilot-study participants were sampled from a fourth year software engineering course offered at the University of Ottawa in January and March 2002. Half of the 32 participants had over one year experience in the software industry.

The official survey, conducted in April 2002, 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 survey data as used in this paper, as well as a more detailed account of the results is available on-line [7]. 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.

1.4 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 may be more appropriate than others, 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.

1.5 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 50 questions of various types including multiple-choice, short answer, ratings, and free-form questions.

The question topics included, among others:

- The role of software team members in the process of writing, maintaining and verifying documentation.
- The participant's personal preference for different types of documentation, and their effectiveness.
- 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. A few replied directly via email.

There were a total of 48 participants that provided responses that 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. Twenty-five 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. Seventeen 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, according to Question 29 of the survey.

Our rationale for the above division is that the proponents of agile techniques promote somewhat different documentation practices from those recommended in conventional software engineering methodologies [1]. [2].

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

- Manager. Twelve individuals that selected manager as one of their current job functions, according to Question 44.
- Developers. Seventeen individuals that are non-managers and selected either senior or junior developer as one of their current job functions, according to Question 44.

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

- Waterfall. Thirteen individuals that selected waterfall as the recommended development process, according to Question 46 of the survey.
- Iterative. Fourteen individuals that are non-waterfall participants and who selected either iterative or incremental development process, according to Question.

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 the participants find most helpful to create, edit, browse and / or generate software documentation.

Table 1 outlines the most frequently cited technologies based on the 41 responses to 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 tools the participant finds the least helpful.

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, there is evidence that 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 removing the need for this type of maintenance altogether.

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 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; such as the artifacts resulting from a low-level design tool. 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 this paper.

3.2 Is Documentation Maintained?

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 the participants from personal experience how long it takes for supporting documentation to be updated when changes in the system occur. The documents we considered include: requirements, specifications, detailed design, low level design, architectural, and 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 when changes occur in a software system?

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

Simlarly, question 20 asked if the participants agreed that documentation is always outdated.

Many participants somewhat agreed (43%) with that statement, and a considerable number of individuals 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, another part of our survey indicates that software quality seems to be improving despite little to no improvement in the quality of software documentation [6].

The evidence that documentation is rarely updated is important from a technology perspective. Since our results imply that 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. The tools must also be efficient from a task perspective, helping users to quickly accomplish what they intended to achieve.

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 documentation useful during inception / construction differs from that which is useful during maintenance and testing.

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

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 the participants if they agreed that most software documents have a finite useful lifetime and should subsequently be discarded or removed.

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	Percentage Disagree (%)	Percentage 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 observe 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 whether, 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 [8] – 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 docu-

mentation. If this hypothesis is true, then a potentially important niche for lightweight documentation tools may emerge. We will further describe the concepts of lightweight documentation, with future work set to determine its relevance to the documentation engineering community.

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 [11]. 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 how often the participant personally consulted the available software documentation between never (1) and always (5).

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: Extent to which the most consulted document type is typically consulted

Participant Category	Document Type most consulted	Mean Consultation
All	Specifications	3.85
Waterfall	Testing / QA	3.88
Iterative	Specifications	4.50
Agile	Specifications	3.47
Conventional	Specifications	4.38
Manager	Requirements	3.60
Developer	Architectural	4.33

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 Relevant document attributes

This section discusses how certain attributes contribute to a document's effectiveness.

Question 9 asked the participants how important particular document attributes contribute to its overall effectiveness. Participants gave rating between 1 (least important) and 5 (most important).

Table 5 lists the attributes considered in Question 9 in descending order based on the attributes perceived contribution to a document's effectiveness.

Table 5: Document attributes and effectiveness

Document Attribute	Mean of Q9	Std. dev.	% Rate 5	% Rate 1 or 2
Content – the document's information	4.85	1.57	85 %	0 %
Up-to-date	4.35	0.89	46 %	0 %
Availability	4.19	0.79	41 %	4 %
Use of examples	4.19	0.85	37 %	4 %
Organization – sections / subsections	3.85	0.64	30 %	4 %
Type – req, spec, design, etc.	3.78	0.63	26 %	11 %
Use of diagrams	3.44	0.60	15 %	22 %
Navigation – quality of internal / external links	3.26	0.44	19 %	33 %
Structure – arrangement of text, diagrams, figures	3.26	0.60	11 %	22 %
Writing Style – sentence / paragraph structure, grammar	3.26	0.67	7 %	19 %
Length – not too long or short	3.15	0.64	7 %	22 %
Spelling and grammar	2.93	0.85	0 %	22 %
Author	2.63	0.41	7 %	48 %
Influence to use it	2.62	0.48	12 %	50 %
Format – pdf,, doc, txt, xml, etc.	2.42	0.58	0 %	54 %

For a more comprehensive analysis of the perceived relevance of certain documentation attributes, please refer to [6]. Specifically relating to documentation engineering, the data suggests that technologies and processes should:

- Focus on content and allow the author to easily create and maintain content rich documents.
- Focus on availability and allow for comprehensive publishing capabilities.
- Focus on examples and allow for better integration of examples within a document.

3.6 Support for document automation

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

Question 24 asked if participants agree that software documentation contains a lot of information that can be extracted directly from the system's source code itself.

Question 30 asked if participants 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. For instance,

- It may be useful for information extraction tools to consider analyzing test source code for the purpose of documentation.
- 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.

3.7 Improving documentation through 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 *software dynamics* (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 participants if they agreed that a tool to track changes in software system for the purpose of document maintenance would be useful.

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

Based on the data from this survey, we believe the following requirements are justified for any technology relating to a documentation tracking mechanism as described above.

First, the technology should complement existing documentation tools to aid tool adoption and project integration. We feel it is important the technology focus on project dynamics (changes to software and related document artifacts) as opposed to documentation maintenance. From Section 3.1, we see that a variety of tools are already employed for documentation tasks and it is unlikely that individuals will adopt new technologies if it means they must abandon current ones.

Second, we suggest the technology support traceability among documents as well as between source code. As such, the technology could apply concepts similar to that of *authoritativeness* in hyperlinked environments [11]. This concept suggests that documents that reference (as well as are referenced by) by the most authoritative sources are also likely to be relevant and worthy of attention. Using proven techniques such as authoritativeness, the technology could improve its ability to rank and recommend relevant documents to its audience.

Third, it is important that the technology be able to report possible discrepancies and suggest the order in which maintenance should occur based on user feedback and the relative importance of the document.

We believe that such a technology would 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 the notion that out-dated documentation can still be a useful [6], one can then base maintenance on user feedback and necessity regarding document inconsistencies as opposed to solely on the fact that a document may be outdated.

3.8 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. Technologies should be easy to use and support the creation of information as opposed to its maintenance.
- Ideas over accuracy. There is evidence to support the claim that documentation functions best as a communication medium (which need not always be accurate or up-to-date to be useful) [6]. Tools should facilitate the communication of ideas, and prompt feedback. For example, a whiteboard and digital camera can create effective 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 [8]. Archiving based on user preferences and usage could reduce the total number of visible documents without actually having to discard any.
- 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.

3.9 Project Size Independence

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

Question 41 asked what for the size of the participants' current project in thousands of lines of code (KLOCS). The available sizes were less than 1, 1-5, 5-20, 20-50, 50-100, over 100 KLOCS or N/A.

Table 6 illustrates the project size distribution for all categories outlined in Section 2.2.

Table 6: Participants Project Size in KLOCs

Participant Category	Percent of projects between 1 and 20 KLOCS	Percent 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

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). 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.

Using Spearman's Rank Correlation [9], the correlation between project size and the individual's software techniques (ranging from highly conventional to highly agile) was very low (-0.09). Similarly, the correlation of project size to the individual's role (ranging from highly managerial to highly developmental) was quite low (0.19).

The low correlation above, and the fair representation of the software categories outlined in Section 2.2 suggest that the results cited in previous 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 in a wide variety of contexts.

Table 7 illustrates the participant's experience in the software field (based on number of years in the industry).

Table 7: 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 8 indicates the current job functions held by the participants. Please note that one individual can have several functions.

Table 8: 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 data above 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 can be helpful in guiding the design of software documentation technologies including tools, notations and methodologies that will more promptly satisfy the needs of real software projects. As well, decision makers will be able to choose more appropriate documentation strategies and technologies based on needs as opposed to generic expectations.

In regards to documentation engineering, the following observations and suggestions can be drawn from this paper:

- 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 and should always serve a purpose. Technologies should enable quick and efficient means to communicate ideas as opposed to providing strict validation and verification rules for building facts.

Learning to cope with the fact that documentation is almost always out-dated and inconsistent, we can then appreciate and utilize it as a tool of communication. Documentation, like communication, can then be judged based on its ability to impart knowledge in its audience; something which need not be up-to-date and consistent (just good enough to serve its purpose) [4].

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.

- Conducting a second, much larger survey on this topic, or a derivative depending on the focus of the research.

- Developing technologies that help track changes in a software system for the purpose of documentation maintenance.
- Developing technologies to rank and recommend documentation based on factors beyond readability, such as authoritative techniques and usage statistics.
- Developing 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 support has 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] Ambler, Scott. *The Fragile Manifesto*, Software Development, August 2002.

[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] Curtis Bill, et al. *A field study of the software design process for large systems*. Communications of the ACM, Volume 31, Number 11, November 1988, p1268 – 1287.

[6] Forward, A. *Software Documentation – Building and Maintaining Artefacts of Communication*. Thesis submission available online at [7]

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

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

[9] Institute of Phonetic Sciences (IFA). http://fonsg3.let.uva.nl/Service/Statistics/RankCorrelation_coefficient.html

[10] 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.

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

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

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

[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] 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.