Software Engineering Documentation Priorities:

An Industrial Study

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 50 individuals in the software field ranging from junior developers to managers and project leaders. One of the goals of this survey was to uncover how software documentation is used in industry and the extent to which, and under what circumstances, documentation can be effective. The data suggest somewhat conflicting views of the importance of documentation maintenance. In particular, participants responded that not-so-up-to-date documents could still be an effective resource. Conversely, the extent to which a document is up-to-date was selected as one of the most important factors in determining its effectiveness. The results suggest that the software industry and academia may overemphasize the importance of document maintenance relative to a software professional's tolerance of out-dated content.

Keywords

Software documentation, 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
- How documentation attributes and artifacts influence its usefulness and relevance to the software team

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 known 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 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)?

One of the large-scale projects was seeking answers about 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 [3]. 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:

- Software engineers, managers and project leaders will learn about how others perceive documentation. Combined with personal experiences this information can help improve documentation in general.
- Software decision makers can use the data to justify modifications to established processes [10] for maintaining and approving documentation.
- Individuals interested in documentation technologies can use the data to design tools that support the documentation process.

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 (and potentially controversial) 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 …

- 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 [3]. A few replied directly via email.

There were a total of 80 responses to the survey. Of these, 50 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 that selected either iterative or incremental as the recommended development process, according to question 46.

3. SURVEY RESULTS

3.1 Documentation Duties

In this section, we discuss who has the responsibility of maintaining as well as validating documentation according to questions 2 and 3 of the survey.

Question 2 asked,

> *In your experience, who has the prime RESPONSIBILITY to CREATE and MAINTAIN the following types of software documentation?*

The participant then selected one answer from the following list

> *Customer(s) / Client(s), Manager / Project Leader, Software Architects / Sr. Developers, Jr. Developers, Technical Writers, or not applicable*

Question 3 asked,

> *In your experience, who has the prime RESPONSIBILITY to VERIFY and VALIDATE the information in the following types of software documentation?*

The participant selected one answer from the same list as described in question 2.

The results from both questions are summarized as follows:

- Requirements are mostly maintained by managers (37%) or clients (33%). Clients (52%) and managers (33%) are most likely responsible to verify and validate these documents.
- Senior developers (48%) and managers (33%) are likely to maintain specification documents. These documents are mostly likely verified and validated by managers (37%) or clients (33%).
- Design documents (architectural, detailed and low-level) are mostly maintained, verified and validated by senior developers, although junior developers are equally likely to have the same responsibility to maintain, verify and validate low-level documentation.
- A considerable percentage of participants selected 'not applicable' when asked to categorize who maintains (23%) or validates (20%) testing and quality documents. This result suggests the documents are maintained, verified and validated by various individuals, or perhaps by no one at all.
- Technical writers are rarely employed to maintain any of these types of documents. This finding questions ideas in [9] that software engineers should not be involved in documentation.

Table 1 and Table 2 illustrate the results of question 2 and 3 based on the categories as outlined in 2.2. Only the findings with at least 50% support of one individual type are listed below. The color scheme that identifies the degree of support is as follows:

- **Bold items** had 75% or more support
- *Italics items* had 60% to 74% support
- Normal items had 50% to 59% support

Table 1: Who should maintain documents?

Participant Category	Req't	Specs	DD	LLD	Arch	QA
All	--	--	Sr. Dev	--	Sr. Dev	--
Agile	--	--	**Sr. Dev**	--	**Sr. Dev**	--
Conventional	--	*Sr. Dev*	**Sr. Dev**	--	**Sr. Dev**	Jr. Dev
Manager	Mngr	Mngr	Sr. Dev	*Sr. Dev*	*Sr. Dev*	Mngr
Developer	--	*Sr. Dev*	*Sr. Dev*	Jr. Dev	**Sr. Dev**	--
Waterfall	--	--	Sr. Dev	--	Sr. Dev	--
Iterative	*Mngr*	--	*Sr. Dev*	--	Sr. Dev	--

Several interesting results include:

- Managers provided the most consistent results. In general, these individuals identified themselves having the most responsibility regarding the documentation process.

- Managers thought they should validate requirements whereas developers believed clients should do it.
- Requirements (req't) and specification (specs) documents are usually maintained by different people from those who verify and validate them.

Table 2: Who should validate documents?

Participant Category	Req't	Specs	DD	LLD	Arch	QA
All	Clients	--	--	--	Sr. Dev	--
Agile	Clients	--	--	--	**Sr. Dev**	--
Conventional	--	--	Sr. Dev	--	Sr. Dev	--
Manager	Mngr	Mngr	--	--	--	Mngr
Developer	*Clients*	Clients	*Sr. Dev*	Sr. Dev	**Sr. Dev**	--
Waterfall	Clients	--	Sr. Dev	Jr. Dev	Sr. Dev	--
Iterative	--	--	Sr. Dev	Sr. Dev	Sr. Dev	--

3.2 Relevant document attributes

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

Question 9 asked the participants

> *How important is each of the following items in helping to create effective software documentation.*

Table 3 lists all attributes from question 9 and highlights the following metrics.

- The attribute's mean and standard deviation
- The percentage of participants rating the attribute 5 (the most important factors)
- The percentage of participants rating the attribute 1 or 2 (the least important factors)

Table 3: Document attributes and effectiveness

Document Attribute	Mean of Q9	Std. dev.	% Rate 5	% Rate 1 or 2
Content	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	3.85	0.64	30	4
Type	3.78	0.63	26	11
Use of diagrams	3.44	0.60	15	22
Navigation	3.26	0.44	19	33
Structure	3.26	0.60	11	22
Writing Style	3.26	0.67	7	19
Length	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	2.42	0.58	0	54

Interesting observations from this data include:

- Content is the most important factor. All document tasks (creation, maintenance, verification, validation) should always keep the target audience in mind. Effective content is the key to effective documentation.
- Extent to which a document is up-to-date is the second most important factor. Although seemingly intuitive and accepted [[1], [4], [6], [7], [11]], the following sections will provide a different interpretation of this result.
- A document's format (.pdf, .doc, .html), its author, influence from management to use it and the quality of spelling and grammar have low correlation with the document's effectiveness.

- The style of writing does not have much impact on effectiveness. This result supports the argument that readability formulas are not an effective metric to determine the usefulness of document [8].

Relating to documentation engineering in the large, documentation technologies should strive to facilitate the above qualities that promote a document's usefulness. In particular, technologies should:

- Focus on content. Allow the author to easily create and maintain content rich documents. This should be the primary intent of most documentation technologies.
- Focus on availability. Allow for larger-scale publishing capabilities to assure the most up-to-date documents are readily available and easily located.
- Focus on examples. Allow for better features to support examples and their integration within a document.

3.3 Documentation Maintenance?

This section illustrates the extent to which documentation is maintained. This information will later be compared to the documentation that is most frequently used, and under what circumstances.

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

Table 4: 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

Based on the participant categorization (see 2.2), several statistically relevant differences occur between agile and conventional participants, as well as those grouped as iterative and waterfall.

Table 5 summarizes the differences between agile and conventional individuals with respect to document maintenance, and Table 6 between iterative and waterfall individuals.

In general, most agile and iterative participants believe documentation is at best updated within a few months of changes to the system. Conversely, conventional and waterfall participants believed documents were maintained within a few months to within a few weeks of system changes.

Table 5: Documentation Update Time (Agile vs. Conventional)

Document Type	Agile		Conventional	
	Mean	St. Dev	Mean	St. Dev
Requirements	2.76	1.30	2.75	1.16
Specifications	3.07	1.16	3.25	1.16
Detailed Design*	2.60	1.06	3.88	1.25
Low Level Design	2.93	1.33	3.57	1.13
Architectural*	2.81	1.05	3.75	1.16
Testing / Quality Documents*	3.31	1.38	4.25	0.89

* Statistically significant differences between the means with 95% confidence

Table 6: Documentation Update Time (Iterative Vs. Waterfall)

Document Type	Iterative		Waterfall	
	Mean	St. Dev	Mean	St. Dev
Requirements*	2.00	1.10	3.25	1.28
Specifications	3.00	1.15	3.63	1.19
Detailed Design**	2.20	1.30	3.50	1.60
Low Level Design**	2.25	1.26	3.83	1.47
Architectural*	2.17	0.75	3.38	1.30
Testing / Quality Documents	3.50	1.29	3.86	1.21

* Statistically significant differences between the means with 95% confidence (** 90% confidence)

Question 20 asked to what degree to you agree with

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

The answers ranged from strongly disagree (a score of 1) to strongly agree (a score of 5).

Many participants (43%) somewhat agreed with that statement, but a considerable number (25%) strongly agreed. In particular, the agile participants were statistically more likely to agree with this statement (mean of 3.83 with standard deviation of 1.20) compared to the conventional participants (mean of 3.08 with standard deviation of 1.29). There is also suggestive, but not statistically significant, evidence that iterative participants are more likely to agree with question 20 then waterfall participants. These results affirm the conclusion from question 6 that iterative and agile participants are less likely to update documentation. As well, the above data support the data from question 6 stating that documentation is rarely updated.

The evidence that agile and iterative individuals work in projects where documentation is less frequently updated and almost always outdated does not imply that these projects are 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. This proposition will become clearer in the following sections.

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

Table 7: 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 3.3.

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

3.5 The Up-to-date Double Standard

This section discusses the importance of keeping software documentation up to date. Two similar questions were posed in the survey with seemingly contradicting results.

Question 21 of the survey asked participants to rate to what degree they agree that

> *Software documentation can be useful even through it might not always be the most up to date (relative the system it documents).*

Table 8 outlines the mean, standard deviation and the percentage of participants who gave this question a score of 5 (the participant *strongly* agrees with the statement).

Table 8: Can out-dated documentation be useful?

Participant Category	Mean of Q21	SD	% Rating 5
All	4.0	0.98	28
Waterfall*	4.1	0.75	38
Iterative*	3.5	0.44	15
Agile	3.9	0.74	28
Conventional	4.1	0.87	29
Manager*	3.3	0.76	8
Developer*	4.1	0.67	35

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

There is overwhelming agreement that out-dated documentation is still quite useful. In all but two categories, the mean was at or above 4.0 (the participant somewhat agrees with the question statement). This observation questions both common intuition as well as past statements that documentation is practically useless unless accurate and kept up to date [[1], [4], [7]].

The conflict might be the assumed association between being up to date and correctness, and inversely not-so-up-to-date with incorrect. Some sources argue that being out-dated implies the information is incorrect and thus not reliable. This unreliability then affects the document's credibility and hence its effectiveness [7].

The above reasoning is based on the notion that documentation must present facts, but some argue its purpose is to convey information [2]. The source code presents the facts and the supporting documents facilitate higher-level interpretation of those facts. A document that instills knowledge in its audience can then be deemed effective, somewhat regardless of its age and the extent to which it is up-to-date [2].

The above data in support of useful out-dated documentation might convince some that the extent to which a document is up-to-date is not that important of a factor to create effective documentation. The survey

data illustrated a somewhat different perspective.

One the factors listed in question 9 (see Section 3.2: Relevant document attributes) was:

Extent to which it is up-to-date

Table 9 outlines the mean, standard deviation and the percentage of participants who rated this item 5 (the item is one of the *most* important factors to determine a documents importance).

Table 9: The importance of up-to-date documents

Participant Category	Mean of Q21	SD	% Rating 5
All	4.3	0.89	46
Waterfall	4.4	1.25	50
Iterative	4.5	3.33	50
Agile	4.3	0.73	44
Conventional	4.3	1.45	43
Manager	4.0	2.23	20
Developer	4.4	1.14	56

The data above illustrates the perceived correlation between a document's maintenance and effectiveness. Alone this result is intuitive, but in combination with the previous results, we present a slightly different interpretation.

Many participants responded that out-dated documentation could be useful. At the same time, many individuals rated the extent to which a document is up to date crucial to determine its usefulness.

This disagreement may influence individuals to over-estimate the importance of the up-datedness of a document relative to several other factors including content, availability and use of examples. Although, we can conclude that while it would be very nice if our documents were up to date, we should not necessarily disregard them or throw them away if they are not up to date. Nor should we attempt to consistently and regularly assure that all documents are up-dated in favor of several important activities; including software construction.

3.6 Documentation in action

This section describes how the participants use and perceive documentation in practice. The survey questions relate to personal exposure to software documentation.

In questions 14 to 17, as well as 34 the participants were asked to rate to what extent they agreed or disagreed with various statements. Ratings were scored as follows: strongly disagree (1), somewhat disagree (2), indifferent (3), somewhat agree (4), and strongly agree (5).

Question 14 states

> *Software documentation is important, but in my organization it is unfortunately not that useful.*

Question 15 states

> *Software documentation that I reference is easy to understand, navigate and cross-reference.*

Question 16 states

> *The language / style of writing in software documentation I reference is brief and to the point.*

Question 17 states

> *When I am working on a software system and require assistance, it is easy to locate the appropriate supporting documentation.*

Question 34 states

> *Software documentation (i.e. the collection of documents describing a particular system) that I reference is poorly organized and difficult to navigate primarily due to the size and number of the documents available.*

In general and as expected, the results provide no overall agreement or disagreement with the above statements.

However, Table 10 compares the results of the above questions between agile and conventional participants.

Table 10: Perception of Documentation (Agile Vs. Conventional)

Question	Agile		Conventional	
	Mean	St. Dev	Mean	St. Dev
Q14	2.35	1.40	2.41	1.54
Q15**	3.56	1.19	3.06	1.39
Q16*	3.56	1.04	2.94	1.03
Q17	2.80	1.12	2.82	1.13
Q34**	2.88	1.19	3.41	1.33

* Statistically significant differences between the means with 95% confidence (** 90% confidence)

The data above suggests that agile participants are statistically more likely to:

- Agree that the documentation they reference is easier to understand, navigate and cross-reference.
- Agree that documentation is brief and to the point.
- Disagree that the collection of documentation is poorly organized and difficult to navigate due to the size and number of available documents.

Despite the fact that documentation is rarely updated (see Section 3.3), agile participants felt that the documentation they reference was brief and to the point, more so than conventional participants. Also, a considerable number of agile participants strongly agree (39%) and somewhat agree (22%) that the documentation in their projects is useful. These results support the claim that documentation need not necessarily be up to date for it to be useful and relevant.

3.7 Software project quality

This section highlights the participants' comparison of current software project quality relative to past projects.

Question 39 asked,

> *Relative to past projects, please compare the quality of the software of your current project. Rate between one (1) for much LOWER quality and five (5) much HIGHER quality.*

Participants made comparisons to software quality based on the following criteria

- # Defects per line of code.
- Team members' pride in project
- Manager's satisfaction with progress
- Customer Satisfaction
- Project delivery on time
- Project delivery on budget

Table 11: Software Project Success Metrics

Quality Metrics	Mean	St. Dev	% Rate 4 or 5
Decreased defects	3.30	1.06	50
Increased pride	3.28	0.98	42
Increased manager Satisfaction	3.16	0.90	29
Increased customer satisfaction	3.59	0.85	57
Projects On time	3.39	1.12	46
Projects On budget	3.26	1.10	46

Most participants cited improvements to software quality based on decreased defects, increased customer satisfaction as well as improved project delivery (both with respect to time and budget).

The belief that software project quality is improving despite the fact that documents are rarely maintained (see Section 3.3) supports the claim of low correlation between successful projects and the extent to which documentation is maintained. If one believes that useful documentation is an important factor to achieve project quality then we can again demonstrate the low correlation between a document's usefulness and the extent to which it is up-to-date.

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

Table 12: 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, which does not necessarily imply 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 41 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 13 illustrates the participant's experience in the software field (based on number of years in the industry).

Table 13: 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 14 highlights the participants current project size. The size is estimated in thousands of lines of source code (KLOCS).

Table 14: Project Sizes in thousands of lines of code (KLOC)

Project Size (KLOCs)	Number of Participants	Percentage
< 1	0	0
1 to 5	1	2
5 to 20	13	28
20 to 50	6	13
50 to 100	5	11
> 100	12	26
Not Applicable	9	20

Table 15 indicates the current job functions held by the participants. Please note that one individual can have several functions.

Table 15: 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).
- 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.

The conclusions cited above will help decision makers choose more appropriate documentation strategies and technologies based on needs as oppose 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 projects should focus more on conveying meaningful and useful knowledge than on precise and accurate information.

5.1 Future Work

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

- How do people use documentation? More in-field research is required to substantiate some of the observations in the paper. What attributes of documentation hinder / facilitate its use?
- How can documentation maintenance be improved? Although maintenance may not be a critical contributor to useful documentation, it would be useful to understand techniques and tools that improve this process.
- What effect does time and change have on a document's relevance? For ex-

ample, as a document ages, its relevance most likely decreases. Why? To what extent? How do external factors affect this decay in relevance?

- How else can we document a system? How effective are these methods with respect to creation, maintenance, use and quality of the content? Research in this area could widen our definition of documentation beyond just documents.

ABOUT THE AUTHORS

Andrew J. Forward is a master's student in Computer Science at the University of Ottawa. He received a Bachelor of Applied Science in Software Engineering (also from the University of Ottawa) in April 2001; the first degree of its kind offered in Canada.

Timothy C. Lethbridge is an associate professor in software engineering at the University of Ottawa. His research interests include software reverse engineering and visualization, software engineering education, and knowledge engineering. He is the lead author of the McGraw-Hill textbook "Object-Oriented Software Engineering: Practical Software Development Using UML and Java". He is also pedagogy co-chair for the ACM/IEEE Computer Society "Computing Curriculum - Software Engineering" project. His web site is http://www.site.uottawa.ca/~tcl.

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.

We would also like to thank Ayana Nurse and Jayne Forward for helping to edit this paper.

REFERENCES

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

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

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

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

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

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

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

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

[9] 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.

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

[11] 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.