Which Documentation For Software Maintenance?

Sergio Cozzetti B. de Souza[1], **Nicolas Anquetil**[1], **Káthia M. de Oliveira**[1]

[1]UCB - Catholic University of Brasilia
SGAN 916 Mômdulo B - Av. W5 Norte
Brasília - DF - 70.790-160, Brazil
{kathia,anquetil}@ucb.br

Abstract

Software engineering has been striving for years to improve the practice of software development and maintenance. Documentation has long been prominent on the list of recommended practices to improve development and help maintenance. Recently however, agile methods started to shake this view, arguing that the goal of the game is to produce software and that documentation is only useful as long as it helps to reach this goal.

On the other hand, in the re-engineering field, people wish they could re-document useful legacy software so that they may continue maintain them or migrate them to new platform.

In these two case, a crucial question arises: "How much documentation is enough?" In this article, we present the results of a survey of software maintainers to try to establish what documentation artifacts are the most important to them.

1. Introduction

Among all the recommended practices in software engineering, software documentation has a special place. It is one of the oldest recommended practices and yet has been, and continue to be, renowned for its absence (e.g. [21]). There is no end to the stories of software systems (particularly legacy software) lacking documentation or with outdated documentation. For years, the importance of documentation has been stressed by educators, processes, quality models, etc. and despite of this we are still discussing why it is not generally created and maintained (e.g. [9]).

The topic gained renewed interest with two recent trends:

- Agile methods question the importance of documentation as a development aid;
- The growing gap between "traditional" (e.g. COBOL) and up-to-date technologies (e.g. OO or web-oriented) increased the pressure to re-document legacy software.

Both issues raise a similar question: What documentation would be most useful to software maintenance?

If they propose a renewed development paradigm, agile methods do not bring significant changes for software maintenance. They do claim that permanent re-factoring turns maintenance into a normal state of the approach, however, they do not explain how such methods would work over extended periods of time, when a development team is sure to disperse with the knowledge it has of the implementation details. Documentation is still a highly relevant artifact of software maintenance.

Legacy software re-documentation tries to remedy the deficiencies of the past in terms of documentation. However, it is a costly activity, difficult to justify to users because it does not bring any visible change for them (at least in the short term).

In this paper we present a survey of software maintainers trying to establish the importance of various documentation artifacts for maintenance. The paper is divided as follows: In Section 2, we review some basic facts about software maintenance and summarize the relevant literature on software documentation; in Section 3, we present the survey we conducted; in Section 4, we analyze and comment its results; Finally, we propose our conclusion and future work in Section 5.

2. Software Documentation and Maintenance

2.1. Software Maintenance

Software maintenance is traditionally defined as any modification made on a software system after its delivery.

Studies show that software maintenance is, by far, the predominant activity in software engineering (90% of the total cost of a typical software [15, 19]). It is needed to keep software systems up-to-date and useful: Any software system reflects (i.e. models) the world within which it operates, when this world changes, the software needs to change accordingly. Lehman's first law of software evolution (law of continuing change, [11]) is that "a program that is used undergoes continual change or becomes progressively less useful". Maintenance is mandatory, one cannot ignore new laws or new functionalities introduced by a concurrent. Programs must also be adapted to new computers (with better performances) or new operational systems.

One of the main problems that affect software maintenance is the lack of up-to-date documentation. Because of this, maintainers must often work from the source code to the exclusion of any other source of information. For example, a study (see [13, p.475], [15, p.35]) reports that from 40% to 60% of the maintenance activity is spent on studying the software to understand it and how the planned modification may be implemented.

To lessen this problem, organizations try to re-document their software systems, but this is a costly operation that would benefit from a clear indication of the software documents to focus on.

2.2. Documentation needs

A software document may be described as any artifact intended to communicate information on the software system [5]. This communication is aimed at human readers.

According to Ambler [1], software documentation attends to three necessities: (i) contractual; (ii) support a software development project by allowing team members to gradually conceive the solution to be implemented, and (iii) allow a software development team to communicate implementation details across time to the maintenance team.

Documentation typically suffers from the following problems: nonexistent or of poor quality [3, 8, 17]; outdated [5, 16, 23, 25, 26]; over abundant and without a definite objective [5, 12]; difficult to access (for example when the documents are scattered on various computers or in different formats: text, diagrams) [25]; lack of interest from the programmers [15, p.45], [7, 24]; and, difficult to standardize, due, for example, to project specificities [7, 14].

Recently, agile methods proposed an approach to software development that mostly eliminated the necessity of documentation as an helper for software development. Using informal communication (between developers and with users), code standardization, or collectivization of the code, agile methods propose to realize the communication necessary to a software development project on an informal level. This ultimately can greatly reduce the need for documentation in the development of software. However, agile methods do not remove the need for documentation as a communication tool through time, that allows developers to communicate important informations on a system to future maintainers.

2.3. Documentation for Maintenance

Better defining what document(s) software maintainers need has already been considered in other studies.

In a workshop organized by Thomas and Tilley at SIGDoc 2001, they state that "no one really knows what sort of documentation is truly useful to software engineers to aid system understanding" [23]. This is precisely the problem we are trying to tackle. Despite this ignorance, many authors have proposed their vision on the document artifacts needed for software maintenance:

- Tilley in [25, 26]) stresses the importance of a document describing the hierarchical architecture of the system.
- Cioch *et al.* [4] differentiate four stages of experience (from new comer, the first day of work; to expert, after some years of work on a system). For each stage, they propose different documents: newcomers need a short general view of the system; apprentices need the system architecture; interns need task oriented documents such as requirement description, process description, examples, step by step instructions; finally, experts need low level documentation as well as requirement description, and design specification.
- Rajlich [18] proposes a re-documentation tool that allows to gather the following information: notes on the application domain, dependencies among classes, detailed description of a class' methods.
- Ambler [1] recommends documenting the design decisions, and a general view of the design: requirements, business rules, architecture, etc.
- Forward and Lethbridge [5], in their survey of managers and developers, found the specification documents to be the most consulted whereas quality and low level documents are the least consulted.
- Grubb and Takang [6, pp.103-106], identify some information needs of maintainers according to their activities. Few specific documents are listed. Managers needs decision support information such as the size of the system, cost of the modification. Analysts need to understand the application domain,

the requirements and have a global view of the system. Designers need architectural understanding (functional components and how they interact) and detailed design information (algorithms, data structures). Finally programmers need a detailed understanding of the source code as well as an higher level view (similar to the architectural view).

- Anquetil *et al.* [2], present a re-documentation tool to partially automate a re-documentation process focusing on the following information: high level view (with description of requirements); data model; cross references between functionalities, functions, and data; business rules; subsystems description and interaction; and comments.

- Finally, according to Teles [22, p.212], the documents that should be generated at the end of an XP project are: stories, tests, data model, class model, business process description, user manual, and project minutes.

All, but one, of these studies are based on the intuition of the authors. Forward and Lethbridge [5] are the only ones to have performed a formal survey as the one we are presenting here. There work differs from our in that they did not consider specifically software maintenance. They also focus on such issues as the best tools to create software documents which is out of the scope of this study.

One conclusion that we may draw from this short review is that system architecture is an important document for software maintenance (cited by Tilley, Cioch, Ambler, Grubb and Takang, and Anquetil *et al.*). We will see that our results do not support this view.

3. SURVEY

To establish the relative importance of various documentation artifacts in helping understand a system, we asked professional software maintainers to rate the importance of different documentation artifacts in helping them understand a system.

In this section we will present the survey we performed (Definition and Planning in Wohlin *et al.* [27] experimentation process). The next section presents the results and there analysis (Operation and Analysis and Interpretation in Wohlin *et al.* experimentation process).

3.1. GOAL DEFINITION

The goal of the survey is to identify the importance of documentation artifacts in helping to understand a system. This would allow focusing the resources of a software development, or software re-documentation, project on the most useful documentation artifacts for maintenance. We are particularly interested in the opinion of actual software maintainers on this issue as opposed to the opinion of academics (for example as presented in the preceding section).

For this we will consider the documentation artifacts typically recommended in known development approaches. We considered two approaches: structured analysis[1] and the Unified Software Development Process.

We formated this goal according to Solingen and Berghout [20] proposal:

> *Analyze* the set of documentation artifacts of a programming paradigm
> *with the purpose of* identify
> *with respect to* importance
> *from the point of view of* software maintenance professionals
> *in the context of* a given programming paradigms.

3.2. PLANNING

The steps in planning a survey are:

Context selection: The survey will target professional software maintainers. The subject will participate off-line, filling a questionnaire that they will return on their own time.

Hypothesis formulation: We are interested in the importance of documentation artifacts in helping to understand a software system. Let us define a metric M_i, which is the mean importance of a documentation artifact i (for a given programming paradigm) according to the opinion of our subjects. M_i, is the mean of all the answers of the subjects, that know artifact i considering the values: "no importance"=1, "little importance"=2, "important"=3, and "very important"=4 (Note: We chose a 4 choices scale in conformance to the suggestion of the ISO/IEC 9126). When a subject did not know a given documentation artifact, his/her answer is not computed in the metric for that artifact. For structured analysis $i \in [1, 24]$ (24 documentation artifacts), for the unified process, $i \in [1, 25]$ (25 documentation artifacts).

The hypotheses we are trying to confirm are:

Null hypothesis H_0^{SA}: For the structured analysis, all documentation artifacts have the same importance for professional maintainers in helping to understand a software system

$$H_0^{SA} : \forall i \in [1, 24], \forall j \in [1, 24] | M_i = M_j$$

[1] It is our experience that, prior to the USDP and the UML, different countries had different "universal" development approaches. In Brazil, all software engineers usually know a technique called "structured analysis", in France it would be the "Merise method", in Germany it appears to be the "V method", etc.

Alternative hypothesis $\mathbf{H_1^{SA}}$: For the structured analysis, documentation artifacts help professional maintainers understand a software system to different degree

$$H_1^{SA} : \exists i \in [1,24], \exists j \in [1,24] | M_i \neq M_j$$

Null hypothesis $\mathbf{H_0^{OO}}$: For the unified process, all documentation artifacts have the same importance for professional maintainers in helping to understand a software system

$$H_0^{UP} : \forall i \in [1,25], \forall j \in [1,25] | M_i = M_j$$

Alternative hypothesis $\mathbf{H_1^{OO}}$: For the unified process (Object-Oriented), documentation artifacts help professional maintainers understand a software system to different degree

$$H_1^{OO} : \exists i \in [1,25], \exists j \in [1,25] | M_i \neq M_j$$

Subject selection: The selection of the subjects will be done by convenience, using the contacts of one of the authors who is a professional software engineer. Participation is voluntary and anonymous basis. The questionnaire is also available on the Internet for any one to answer it.

Instrumentation: The hypotheses will be tested through a questionnaire distributed to the subjects on paper and available on the internet. The questionnaire is composed of two parts. The first part is a characterization of the subject: Position (manager, analyst, or programmer); Experience in maintenance (1-3 years, 3-5 years, 5-10 years, >10 years); Experience in number of systems already maintained (1 to 5, 6 to 10, 11 to 20, more than 20); and, Known approaches (structured analysis, object-orientation[2]). The subject could also indicate his-her email (optional) in case s-he wished to receive the results of the survey.

The second part of the questionnaire (see Figure 1) asked the subjects to answer the following question for a list of documentation artifacts: "Based on your practical experience, indicate what importance each documentation artifact have, in the activity of understanding a software to be maintained". Four levels of importance where proposed (see above): no importance, little importance, important, and very important. The subjects could also indicate that they did not know the artifact.

The documentation artifacts were divided by activities of a typical development process, discriminating for each activity, artifacts specific to the structured analysis (e.g. context diagram), the unified process (based on the RUP, e.g. use case diagram), or both (e.g. Entity-Relationship Model). The complete list of 34 artifacts, as they were presented in the questionnaire is the following:

Requirement elicitation: Structured analysis documents are requirements list, context diagram, requirement description; the unified process documents are vision document, use case diagram; Documents common to both paradigms are conceptual data model, glossary.

Analysis: Structured analysis documents are functions derived from the requirements, hierarchical function diagram, data flow diagram; the unified process documents are use cases specifications, class diagram, activity diagram, sequence diagram, state diagram; Documents common to both paradigms are non functional prototype, logical data diagram (MER), data dictionary.

Design: Structured analysis documents are architectural model, general transaction diagram, components specification; the unified process documents are collaboration diagram, components diagram, distribution diagram; Documents common to both paradigms are physical data model, functional prototype.

Coding: Documents common to both paradigms are comments in source code, source code.

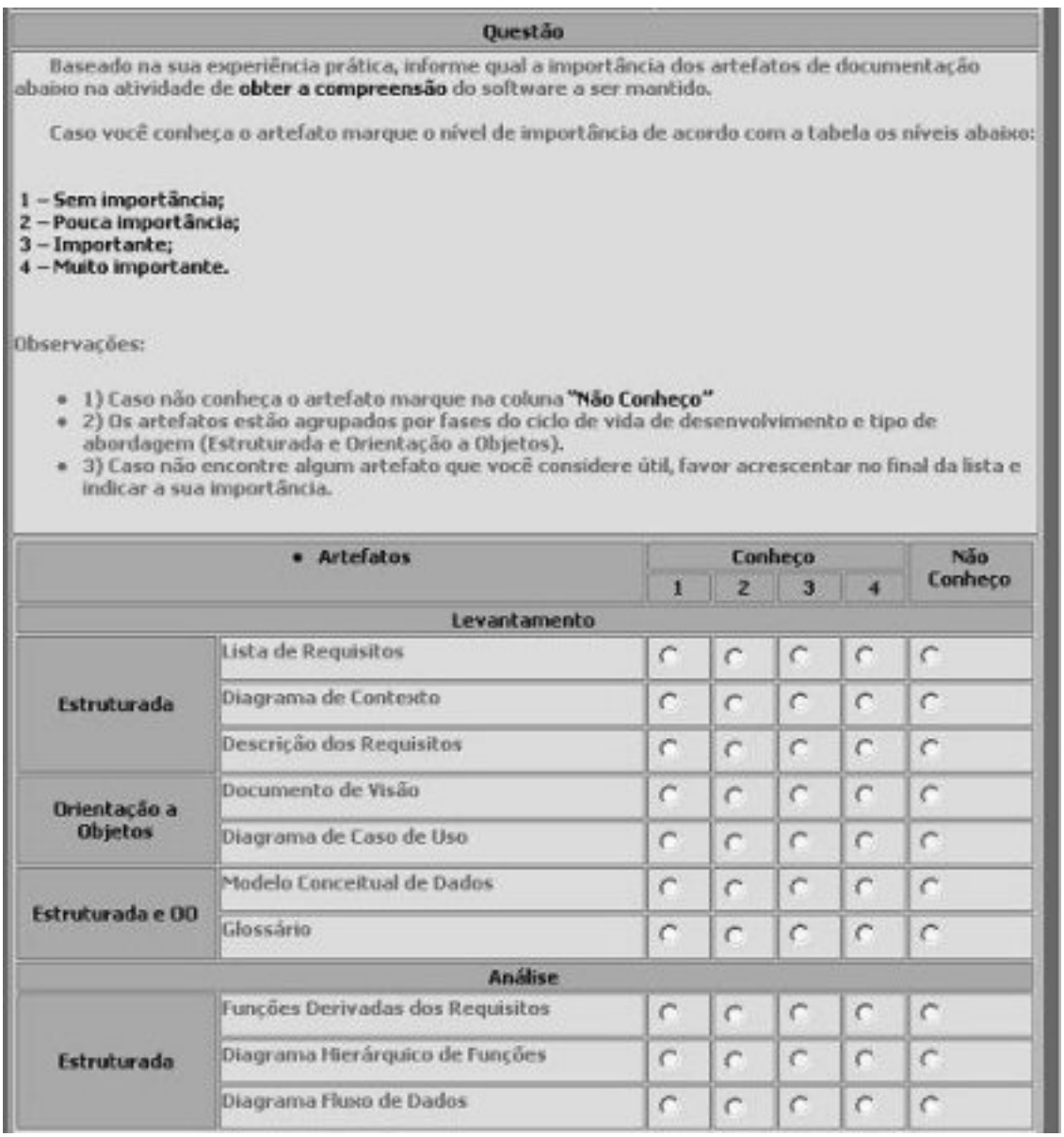
Test: Documents common to both paradigms are unitary test plan, system test plan, acceptance test plan.

Transition: Documents common to both paradigms are data migration plan, transition plan, user manual.

Validity evaluation: Different issue may represent possible validity problems.

- In the construction of the questionnaire, the list of documentation artifacts should contain all documentation artifacts that one could reasonably expect to be using. As explained, we used the artifacts recommended by two well documented methodologies (Structured analysis and Unified Software Developement Process), thus we do not believe that this list represent any validity problem.
- The subjects were answering the questionnaire without our supervision and there is a risk that they did not know what a particular documentation artifact meant, or hada different understanding of what a document is (i.e.believing

[2] i.e. the unified process



Questão

Baseado na sua experiência prática, informe qual a importância dos artefatos de documentação abaixo na atividade de **obter a compreensão** do software a ser mantido.

Caso você conheça o artefato marque o nível de importância de acordo com a tabela os níveis abaixo:

1 – Sem importância;
2 – Pouca importância;
3 – Importante;
4 – Muito importante.

Observações:

- 1) Caso não conheça o artefato marque na coluna **"Não Conheço"**
- 2) Os artefatos estão agrupados por fases do ciclo de vida de desenvolvimento e tipo de abordagem (Estruturada e Orientação a Objetos).
- 3) Caso não encontre algum artefato que você considere útil, favor acrescentar no final da lista e indicar a sua importância.

• Artefatos		Conheço				Não Conheço
		1	2	3	4	
Levantamento						
Estruturada	Lista de Requisitos	○	○	○	○	○
	Diagrama de Contexto	○	○	○	○	○
	Descrição dos Requisitos	○	○	○	○	○
Orientação a Objetos	Documento de Visão	○	○	○	○	○
	Diagrama de Caso de Uso	○	○	○	○	○
Estruturada e OO	Modelo Conceitual de Dados	○	○	○	○	○
	Glossário	○	○	○	○	○
Análise						
Estruturada	Funções Derivadas dos Requisitos	○	○	○	○	○
	Diagrama Hierárquico de Funções	○	○	○	○	○
	Diagrama Fluxo de Dados	○	○	○	○	○

Figure 1. Part of the questionnaire used in the survey. The top half part (lighter gray) states the question, the bottom half part (darker gray) is the list of artifacts and the placeholders to mark their importance.

they know waht it is when they don't). Because the two methodologies chosen are well known, we believe this risk was minimal. We also provided the possibility to indicate one did not know some particular artifact. To keep the material light, we did not provide any definition or other explanation on the documentation artifacts.

- Again because the subjects were answering the questionnaire without our supervision, there is a risk that they did not understand the question asked. We first applied the questionnaire to two professional software engineers (that did not participate further in the survey) to ensure that they understood the question and what they were expected to do. This test helped us to improve the phrasing of the question.
- We are assuming that the set of subjects that responded the questionnaire is representative of the general software maintainer population. An analysis of the characteristics of the set of subjects will be presented in Section 4. We have however no means to formally verify this assumption as we are not aware of any existing characterization of the general software maintainer population.
- The experience of the maintainers or their function could have an impact on the results. However, we will see that the experience and the function of the subjects are evenly distributed.

4. SURVEYS RESULTS

4.1. RAW RESULTS

A total of 76 professional maintainers answered the survey over a period of 6 months (July to December 2004). As part of the answers were submitted by internet, we don't have an exact picture of the geografical distribution of the subjects. From the informations we have (answers submitted on paper and emails collected on some electronic answers), we may infer that the vast majority of the subjects, if not all, are from Brazil, located mainly in the cities of SÃčo Paulo and Brasilia, two of the larger cities of Brazil in terms of TI industry. They proved to be a heterogeneous sample population that we believe to be representative. However, for lack of data on the general software maintainer population, it is difficult to evaluate this representativeness.

Position: total=76, manager=20 (26%), analyst=48 (63%), programmer=5 (7%), and 3 consultants (4%). This appear to show a population biased toward the "higher levels" of the profession. However, one must also consider that many programmers are actually called analyst / programmer, or sometimes junior analyst, which may distort the characterization. (See Figure 2, left part).

Known approach: total=76, structured=22 (29%), OO=6 (8%), both=48 (63%). We were surprised by the high quantity of software engineers who declared knowing object-orientation (63+8=71%), but it may only reflects the popularity of the RUP and UML rather than actual OO programming. (See Figure 2, right part).

Experience (years): total=76, 1-3 years=17 (22%), 3-5 years=19 (25%), 5-10 years=17 (22%), >10 years=23 (30%). The experience is evenly distributed. (See Figure 3, left part).

Experience (number of systems maintained): total=76, 1-5=26 (34%), 6-10=15 (20%), 11-20=15 (20%), >20=20 (26%). Again the experience in number of system maintained is evenly distributed. (See Figure 3, right part).

Tables 1 and 2 give the result of the survey for structured analysis and the unified process. The results are ranked in decreasing order of importance (based on the mean importance as defined by the metric M_i defined in Section 3.2). For each artifact, the number of subject considered includes only those that know this artifact. The value of the metric is not given in these tables for lack of space, but it may easily be computed from the numbers presented and may also be found in two other tables (4 and 5) presented later. For example for the Source code artifact, the mean importance for structured analysis is $M = (5 \times 3 + 63 \times 4)/68 = 3,93$, and for the Unified Process $M = (3 \times 3 + 50 \times 4)/53 = 3,94$.

4.2. STATISTICAL ANALYSIS OF THE RESULTS

Results were analyzed using two statistical testes: ANOVA and Tukey HSD. The first allow to test the difference between means, it will be used to refute our null hypotheses. The second allow to find the significant difference between means, it will be used to define groups of document artifacts which are statistically equivalent (of the same importance).

The values for the ANOVA are given in Table 3. In both case, the p-value is below 0,01 which allows us to reject both null hypotheses ($H^{SA}0$ and $H^{OO}0$) with a significance level of 1%. Therefore, we must accept the alternative hypothesis that, in both programming paradigm, there are statistically significant differences between the degrees to which different documentation artifacts help professional software maintainers in understanding a system.

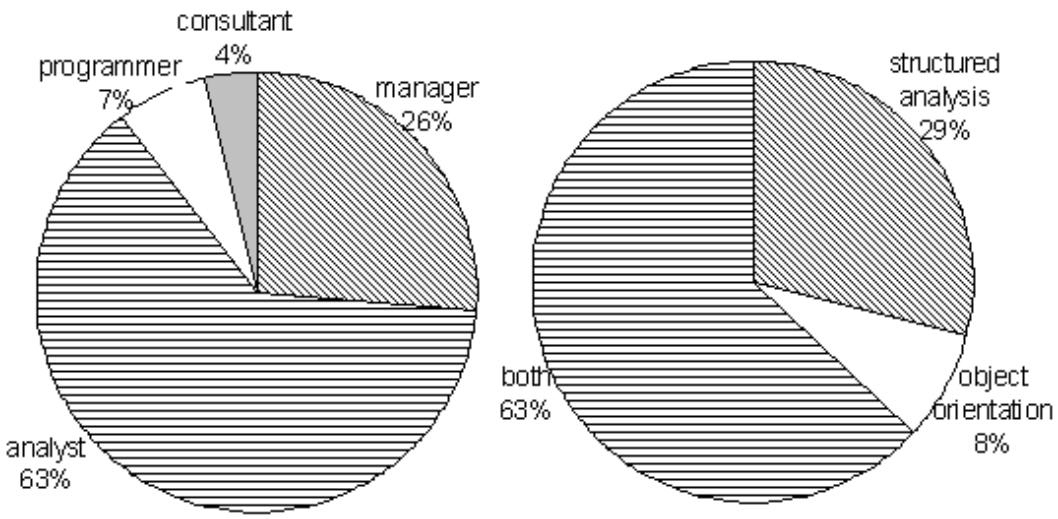


Figure 2. Raw results for the position (left part) and approach known (right part) by the subjects of our survey.

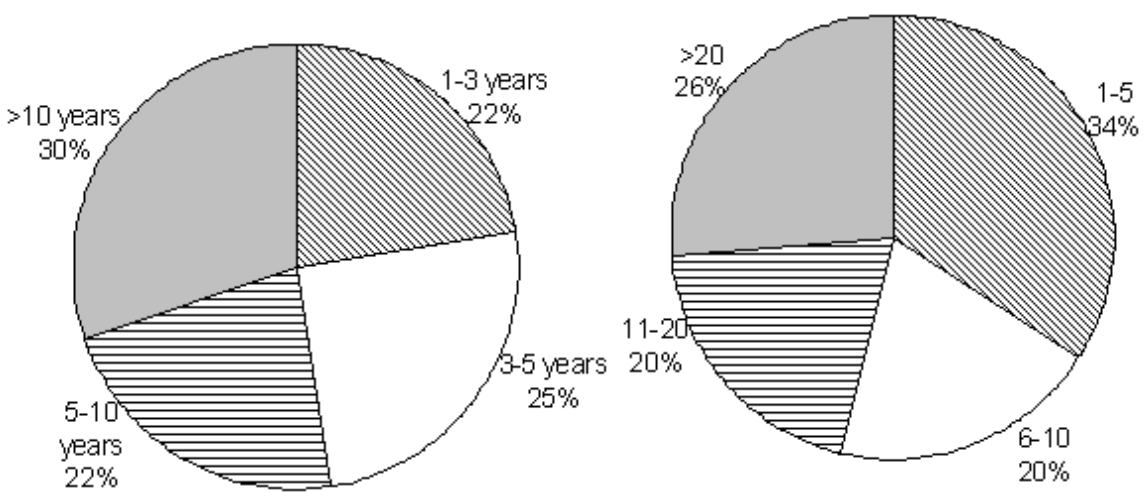


Figure 3. Raw results for the maintenance experience in years (left part) and number of system maintained (right part) by the subjects of our survey.

Table 1. Importance of documentation artifacts for the structured analysis paradigm (according to 70 software maintainers).

Structured analysis artifact	Important				Total	Does not know
	no	little	yes	very		
Source code	0	0	5	63	68	2
Comments	0	4	11	54	69	1
Logical data model (MER)	0	3	14	50	67	3
Physical data model	0	1	24	42	67	3
Requirement description	3	7	18	41	69	1
Data dictionary	1	10	24	31	66	4
Requirement list	6	9	17	36	68	2
Acceptance test plan	6	8	16	34	64	6
Conceptual data model	4	7	25	29	65	5
System test plan	4	10	23	29	66	4
Implantation plan	5	7	27	28	67	3
User manual	6	8	23	29	66	4
Unitary test plan	5	12	22	25	64	6
Data flow diagram	5	11	29	23	68	2
Data migration plan	6	10	25	23	64	6
Component specification	4	10	32	19	65	5
Functional prototype	7	12	24	21	64	6
Architectural model	5	15	26	18	64	6
Hierarchical function diagram	5	15	30	15	65	5
Glossary	4	21	26	15	66	4
General transaction diagram	5	14	25	11	55	15
Context diagram	4	25	21	17	67	3
Functions derived from requirements	4	17	21	12	54	16
Non functional prototype	8	13	29	12	62	8

Table 2. Importance of documentation artifacts for the Unified process (according to 54 software maintainers).

Unified process artifacts	Important				Total	Does not know
	no	little	yes	very		
Source code	0	0	3	50	53	1
Comments	0	4	9	41	54	0
Logical data model (MER)	0	3	12	38	53	1
Class diagram	0	2	18	33	53	1
Physical data model	0	2	19	32	53	1
Use case diagram	0	5	17	31	53	1
Use case specification	1	3	21	26	51	3
Data dictionary	1	8	19	24	52	2
Acceptance test plan	2	8	15	25	50	4
Conceptual data model	2	5	22	22	51	3
Sequence diagram	0	5	30	18	53	1
Implantation plan	2	9	18	23	52	2
System test plan	1	9	22	20	52	2
User manual	3	8	19	23	53	1
Unitary test plan	1	10	19	19	49	5
Data migration plan	3	9	18	19	49	5
Activity diagram	3	7	27	15	52	2
Vision document	2	10	23	13	48	6
Glossary	3	13	25	11	52	2
Functional prototype	6	13	19	14	52	2
Component diagram	5	9	28	8	50	4
Non functional prototype	6	10	22	11	49	5
State diagram	6	14	23	7	50	4
Distribution diagram	5	15	26	3	49	5
Collaboration diagram	6	14	28	1	49	5

Table 3. ANOVA for the Structured Analysis survey and the unified process survey

ANOVA — Structured Analysis					
	sum of squares	degree of freedom	Mean square	F	p-value
Overall	169,224	23	7,358	10,184	0,000
Within groups	1109,676	1536	0,722		
Total	1278,900	1559			
ANOVA — Unified Process					
	sum of squares	degree of freedom	Mean square	F	p-value
Overall	174,141	24	7,256	12,268	0,000
Within groups	743,465	1257	0,591		
Total	917,607	1281			

The ANOVA test only tells us that the documentation artifact are more or less important, but it does not tell which artifacts may be considered of the same importance or not. For this we used the Tukey HSD test.

Tukey HSD allows to define homogeneous subsets for which the difference of the documentation artifacts are not statistically significant. For example, in Table 4, the four structured analysis artifacts: Source code, Comments, Logical data model, and Physical data model, all belong to the seventh subset. This means they have statistically the same importance in our survey. This is a statistical analysis, a more intuitive analysis from the point of view of software engineering will be proposed in the next section. Similarly, Table 5 gives the homogeneous subsets for the unified process documentation artifacts.

4.3. Software engineering interpretation

Our goal was to help software maintainers decide what documentation artifacts are more important to help understand a system. To better fulfill this goal, we now propose a more intuitive interpretation of the results:

- One difference between the two programming paradigms is that for the unified process, the topmost subsets are more inclusive (larger) and the last one more exclusive (smaller) whereas it is the opposite for structured analysis. For example the two higher subsets for structured analysis include only 6 artifacts and all the rest (18 artifacts) belongs to the last subset. For the unified process, on the other hand, the first two subsets hold 15 artifacts and the last one only 9 (one artifact does not belong either to the first two subsets, or to the last one).

- Apart from this first case, in general, the two programming paradigms gave similar results and the other conclusions apply to both.

- It is not a surprise to see the source code and the comments (in the source code) appearing at the top of the most important documentation artifacts to help understand a system.

- If we look at the higher subset in both programming paradigms, we see that data models are also very important. For the structured analysis paradigm they are the logical and physical data model, for the Unified process, it also includes the class diagram.

- The second topmost subset for structured analysis includes the Data dictionary. Which is related to the data model. The second topmost subset for the unified process also includes the Data dictionary, but we may desconsider this subset because of the inclusive nature of the subsets (see second point of this discussion.)

- Another high-level view of the system which is favored by the maintainers is a documentation of the requirements. The Use case diagram and the Use case specification appear in the topmost subset for the unified process and Requirements description appears in the second topmost subset for structured analysis.

- On the other end of the spectrum, prototypes (either functional or not), and abstract view of the system (e.g. Architectural model, Vision document, Context diagram,) ranked low and appear in the last subset. In the case of the Architectural model this was a surprise as it is an artifact generally well considered in the literature (see Section 2).

Overall, the opinion of software maintainers, is that the most important documentation artifacts, to help understanding a system, are: the source code (with comments), a data model (whether it is logical, physical or class diagram), and specification of the requirements. This agrees with the opinion of various authors as cited in Section 2, the data model is cited in [2, 22] and the requirement specification in [1, 2, 5, 6, 22].

One of the interest of this study is that appart from the most important documentation artifacts, the survey also identifies the artifacts considered less important by the software maintainers and that should be the first candidates to consider when trying to simplify a process. These artifacts are system prototype, architectural view, and many artifacts of the design activity (e.g. component diagram, component specification). This result does not agree with the opinion of other authors as already highlighted.

Possible extensions to this study include analysing the possible impact of the position or experience of the maintainer in the evaluation of the importance of the documentation artifacts. We do not have enough data to draw any statistically significant conclusion, however, from our data, there seems to be a difference in opinion depending on the position of the software maintainer. For example, programmers tend to value more any documentation artifact than analysts. Also, and ironically, programmer tend to value more analysis documentation artifact than the analysts, whereas analysts value more the source code and comments than programmers. We could not identify any clear trend associated to the experience of the software maintainers.

5. Conclusion and Future work

Software documentation has long been a much discussed topic in software engineering. Although it has always been heralded as an important aid to software de-

Table 4. Homogeneous subset of document artifact for Structured analysis according to the Tukey HSD test. Alpha=0,05. We used a harmonic average of 64,774 for the size of the samples (N) as the sizes are different.

			Homogeneous subsets						
Artifact	N	Mean	1	2	3	4	5	6	7
Source code	68	3,93							x
Comments	69	3,72						x	x
Logical data model (MER)	67	3,70					x	x	x
Physical data model	67	3,61				x	x	x	x
Requirement description	69	3,41			x	x	x	x	
Data dictionary	66	3,29		x	x	x	x	x	
Requirement list	68	3,22	x	x	x	x	x		
Acceptance test plan	64	3,22	x	x	x	x	x		
Conceptual data model	65	3,22	x	x	x	x	x		
System test plan	66	3,17	x	x	x	x			
Implantation plan	67	3,16	x	x	x	x			
User manual	66	3,14	x	x	x	x			
Unitary test plan	64	3,05	x	x	x				
Data flow diagram	68	3,03	x	x	x				
Data migration plan	64	3,02	x	x	x				
Component specification	65	3,02	x	x	x				
Functional prototype	64	2,92	x	x	x				
Architectural model	64	2,89	x	x	x				
Hierarchical function diagram	65	2,85	x	x					
Glossary	66	2,79	x	x					
General transaction diagram	55	2,76	x	x					
Context diagram	67	2,76	x	x					
Functions derived from requirements	54	2,76	x	x					
Non functional prototype	62	2,73	x						

Table 5. Homogeneous subset of document artifact for the unified process according to the Tukey HSD test. Alpha=0,05. We used a harmonic average of 51,222 for the size of the samples (N) as the sizes are different.

			Homogeneous subsets							
Artifact	N	Mean	1	2	3	4	5	6	7	8
Source code	53	3,94								x
Comments	54	3,69							x	x
Logical data model (MER)	53	3,66							x	x
Class diagram	53	3,58						x	x	x
Physical data model	53	3,57						x	x	x
Use case diagram	53	3,49					x	x	x	x
Use case specification	51	3,41					x	x	x	x
Data dictionary	52	3,27				x	x	x	x	
Acceptance test plan	50	3,26				x	x	x	x	
Conceptual data model	51	3,25				x	x	x	x	
Sequence diagram	53	3,25				x	x	x	x	
Implantation plan	52	3,19				x	x	x	x	
System test plan	52	3,17			x	x	x	x	x	
User manual	53	3,17			x	x	x	x	x	
Unitary test plan	49	3,14			x	x	x	x	x	
Data migration plan	49	3,08		x	x	x	x	x		
Activity diagram	52	3,04	x	x	x	x	x	x		
Vision document	48	2,98	x	x	x	x	x			
Glossary	52	2,85	x	x	x	x				
Functional prototype	52	2,79	x	x	x	x				
Component diagram	50	2,78	x	x	x	x				
Non functional prototype	49	2,78	x	x	x	x				
State diagram	50	2,62	x	x	x					
Distribution diagram	49	2,55	x	x						
Collaboration diagram	49	2,49	x							

velopment and maintenance, it is notoriously absent or out-dated in many legacy software.

Recently, agile methods have shaken a bit the traditional view of software documentation, proposing a development model that relies more on informal communication than on documentation. We explained, however, that this model is not suited to software maintenance which still has great need for documentation.

The question arose then to identify the documentation artifacts most important to help software maintainers. We conducted a survey among software maintainers to try to settle this issue. In this survey, professional maintainers were asked what artifacts they though important or not in helping understand the system to maintain. The subject population was heterogeneous and, although we have no comparative data to prove it, seems representative of the broad community of software maintainers.

The survey confirmed that source code and comments are very important documentation artifact. Data model and requirement specification appear also at the top of the list. Surprisingly, and contrary to what we found in the literature, architectural models and other abstract view of the system are not judged important. This could indicate that such documentation artifacts are used once to have a global understanding of the system and never looked at again after. This explanation would make a difference between quantity and quality: the architectural model is little used, but could, nevertheless, be important.

Further research is needed in the same direction, for example Cioch *et al.* [4] states that different stages of experience have different documentation requirement. A similar experience could be setup to correlate the importance of document artifacts with the experience of the maintainer. Similarly, Grubb and Takang [6] differentiate the activities of the software engineers. Again this could be verified with a similar experiment.

REFERENCES

[1] S. W. Ambler. Agile documentation. available on the internet at: http://www.agilemodeling.com/essays/ agileDocumentation.htm, 2001-2005. Last accessed on May 27, 2005.

[2] N. Anquetil, K. M. Oliveira, A. G. dos Santos, P. C. da Silva jr., L. C. de Araujo jr., and S. D. Vieira. A tool to automate re-documentation. In *Forum of the CAISE, Conference on Advanced Information Systems Engineering (CAiSE'05)*, jun. 15 2005. accepted for publication.

[3] L. C. Briand. Software documentation: How much is enough. In *Proceedings of the Seventh European Conference on Software Maintenance and Reengineering (CSMR'03)*, pages 13–17. IEEE, IEEE Comp. Soc. Press, March 26 - 28 2003.

[4] F. A. Cioch and M. Palazzolo. A documentation suite for maintenance programmers. In *Proceedings of the 1996 International Conference on Software Maintenance (ICSM'96)*, pages 286–95. IEEE, IEEE Comp. Soc. Press, Nov 1996.

[5] A. Forward and T. C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *DocEng '02: Proceedings of the 2002 ACM symposium on Document engineering*, pages 26–33, New York, NY, USA, 2002. ACM Press.

[6] P. Grubb and A. Takang. *Software Maintenance: Concepts and Practice*. World Scientific Publishing Co., Singapore, 2nd edition, 2003.

[7] HCI. What to put in software maintenance documentation. Available on the Internet at: http://www.hci.com.au/hcisite2/journal/ What to put in software maintenance documentation.htm, 2001–2002. Last accessed on May 27, 2005.

[8] S. Huang and S. Tilley. Towards a documentation maturity model. In *SIGDOC '03: Proceedings of the 21st annual international conference on Documentation*, pages 93–99, New York, NY, USA, 2003. ACM Press.

[9] M. Kajko-Mattsson. The state of documentation practice within corrective maintenance. In *Proceedings of the International Conference on Software Maintenance (ICSM'01)*, pages 354–363. IEEE, IEEE Comp. Soc. Press, Nov. 07-09 2001.

[10] B. A. Kitchenham, G. H. Travassos, A. von Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, and H. Yang. Towards an ontology of software maintenance. *Journal of Software Maintenance: Research and Practice*, 11:365–389, 1999.

[11] M. Lehman. Programs, life cycles and the laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–76, sept. 1980.

[12] M. Lindvall, V. R. Basili, B. W. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. A. Williams, and M. V. Zelkowitz. Empirical findings in agile methods. In *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002*, pages 197–207, London, UK, 2002. Springer-Verlag.

[13] S. L. Pfleeger. *Software Engineering: Theory and Practice*. Prentice Hall, 2nd edition, 2001.

[14] V. Phoha. A standard for software documentation. *Computer*, 30(10):97–98, Oct. 1997.

[15] T. M. Pigoski. *Practical Software Maintenance: Best Practices for Software Investment*. John Wiley & Sons, Inc., 1996.

[16] C. J. Poole, T. Murphy, J. W. Huisman, and A. Higgins. Extreme maintenance. In *International Conference on Software Maintenance, ICSM'01*, pages 301–10. IEEE, IEEE Comp. Soc. Press, Nov. 2001.

[17] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 5th edition, 2001.

[18] V. Rajlich. Incremental redocumentation using the web. *IEEE Software*, 17(5):102–6, Sep 2000.

[19] R. C. Seacord, D. plakosh, and G. A. Lewis. *Modernizing Legacy Systems – Software technologies, engineering processes, and business practices*. Addison-Wesley, 2003.

[20] R. van Solingen and E. Berghout. *The Goal/Question/Metric Method: A practical guide for quality improvement of software development*. McGraw-Hill, 1999.

[21] M. J. Sousa. A survey on the software maintenance process. In *International Conference on Software Maintenance, ICSM'98*, pages 265–74. IEEE, IEEE Comp. Soc. Press, Mar. 1998.

[22] V. M. Teles. *Extreme Programming*. Novatec Editora Ltda, Rua cons. Moreira de Barros, 1084, conj. 01, São Paulo, SP, 02018-012, Brazil, 2004. ISBN: 85-7522-047-0.

[23] B. Thomas and S. Tilley. Documentation for software engineers: what is needed to aid system understanding? In *SIGDOC '01: Proceedings of the 19th annual international conference on Computer documentation*, pages 235–236, New York, NY, USA, 2001. ACM Press.

[24] S. Tilley and H. Müller. Info: a simple document annotation facility. In *SIGDOC '91: Proceedings of the 9th annual international conference on Systems documentation*, pages 30–36, New York, NY, USA, 1991. ACM Press.

[25] S. R. Tilley. Documenting-in-the-large vs. documenting-in-the-small. In *Proceedings of CASCON'93*, pages 1083–90. IBM Centre for Advanced Studies, Oct. 1993.

[26] S. R. Tilley, H. A. Müeller, and M. A. Orgun. Documenting software systems with views. In *Proceedings of the 10th International Conference on Systems Documentation, SIGDOC'92*, pages 211–19. ACM, ACM Press, Oct 1992.

[27] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, 2000. ISBN: 0-7923-8682-5.