Software Documentation Strategy for Existing Web Systems

A case study

Mikael Olsson
2016

Bachelor of Science
Information Systems Science

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Abstract

What kind of software documentation a system needs and how much documentation is necessary are questions that have a lot of different answers depending on a number of variables. Traditional software development methodologies claim that software needs a lot of documentation and more agile approaches claim that it is better to write less documentation since most documentation is never used. According to studies a ratio of 11% of software projects costs are spent on documentation alone. The purpose of this study is to create a cost efficient software documentation strategy for an existing web system with a focus on deciding what information is relevant to document in order to keep a high ROI. This study was conducted as a single case study and made in collaboration with a company. The data collection was done by interviewing key people working in the system and doing participants observations. The result shows that information documented in a high level is what is most needed. Artifacts relevant to document are the source code, requirements of updates, functional tests, high-level architecture, reference manual and an end-user manual. The result also shows that new processes need to be implemented for the documentation strategy to be efficient. Recommendation for further research is to create a method of how to calculate the ROI for software documentation based on a number of organizational variables.

Keywords: Software documentation, software documentation details, software documentation strategy
Preface

This thesis of 15 ECTS was submitted to fulfill Bachelor of Science in Information Systems (180 ECTS) at the Department of Computer Science, Electrical and Space Engineering at Luleå University of Technology. This study has been under the supervision of Ali Ismail Awad, Senior Lecturer, Luleå University of Technology at the Department of Computer Science, Electrical and Space Engineering.

This study has been made in collaboration with a company and with help of external supervision from employees at the company.

I want to thank the company studied for the opportunity and confidence in letting me do this research, and for the support and the welcome I received during my stay at the office. A special thank you to the IT/R&D department for giving me their time to be able to conduct this study. I also want to thank my supervisor and my opponents for their valuable comments during this research.
# Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Git project</td>
<td>Project stored in a source code management system</td>
</tr>
<tr>
<td>IT service desk</td>
<td>Internal department that handles problems within the systems</td>
</tr>
<tr>
<td>MVC</td>
<td>Framework for building web applications</td>
</tr>
<tr>
<td>NDA</td>
<td>Non-disclosure agreement, agreement of how to handle confidential information</td>
</tr>
<tr>
<td>ORM</td>
<td>A technique for accessing relationship databases with object oriented language</td>
</tr>
<tr>
<td>phpDocs</td>
<td>A tool for organizing and documenting PHP source code</td>
</tr>
<tr>
<td>ROI</td>
<td>Return of investment</td>
</tr>
<tr>
<td>TCO</td>
<td>Total cost of ownership</td>
</tr>
</tbody>
</table>
# Table of Content

Chapter 1 ................................................................. 7
Introduction .................................................................. 7
  1.1. Problem Definition.................................................. 8
  1.2. Research Question.................................................. 8
  1.3. Delimitations and Limitations ................................... 8
  1.4. Research Purpose.................................................. 8
  1.5. Expected Contribution From This Study...................... 8
Chapter 2 ...................................................................... 9
Theoretical Background.................................................. 9
  2.1. Agile Documentation................................................ 9
  2.2. Types of Documentation......................................... 9
    2.2.1. User Documentation......................................... 10
    2.2.2. System Documentation...................................... 11
  2.3. Amount of Documentation....................................... 12
  2.4. Creation of Documentation...................................... 13
  2.5. Writing Guidelines............................................... 14
  2.6. Documentation Review.......................................... 14
  2.7. Kruchten 4+1 Model.............................................. 15
Chapter 3 ..................................................................... 16
Literature Review.......................................................... 16
Chapter 4 ..................................................................... 18
Research Methodology................................................... 18
  4.1. Research Method................................................... 18
  4.2. Research Design.................................................... 19
  4.3. Preparation.......................................................... 19
  4.4. Collect ............................................................... 19
  4.5. Analyze ............................................................... 20
  4.6. Share ................................................................. 20
Chapter 5 ..................................................................... 22
Data Collection and Analysis........................................... 22
  5.1. Methodology for Primary Research............................ 22
  5.2. Analysis of the Interviews........................................ 22
  5.3. Analysis of the Observations.................................... 25
  5.4. Documentation Needed.......................................... 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.1. Documentation for Developers</td>
<td>26</td>
</tr>
<tr>
<td>5.4.2. Documentation for Server Administrator</td>
<td>27</td>
</tr>
<tr>
<td>5.4.3. Documentation for IT-service desk</td>
<td>28</td>
</tr>
<tr>
<td>5.4.4. Documentation for Testers</td>
<td>28</td>
</tr>
<tr>
<td>5.4.5. Documentation for End Users</td>
<td>29</td>
</tr>
<tr>
<td>5.4.6. Documentation for Documentation Writers</td>
<td>29</td>
</tr>
<tr>
<td>5.5. Documentation of Updates Workflow</td>
<td>29</td>
</tr>
<tr>
<td>5.6. Process Standards</td>
<td>30</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>32</td>
</tr>
<tr>
<td>Conclusions</td>
<td>32</td>
</tr>
<tr>
<td>6.1. Overview</td>
<td>32</td>
</tr>
<tr>
<td>6.2. Conclusion from this Study</td>
<td>32</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>34</td>
</tr>
<tr>
<td>Reflections and Suggestions for Future Research</td>
<td>34</td>
</tr>
<tr>
<td>References</td>
<td>35</td>
</tr>
<tr>
<td>Appendix</td>
<td>37</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1. Different type of user documentation, their scope and target readers (Sommerville, 2011)

Figure 2. Relationship between usefulness of documentation and amount of documentation (Ruping, 2003)

Figure 3. Relationship between total effort and time in documentation late strategy and traditional documentation approach (Agile Modeling, n.d.)

Figure 4. Relationship between total effort and time in continuously documentation strategy and a traditional documentation approach (Agile Modeling, n.d.)

Figure 5. 4+1 model with mapping to UML (Muchandi, 2007)

Figure 6. Case study workflow and how it was implemented in this study

Figure 7. Workflow over the process for documenting updates

List of Tables

Table 1. Summary of respondents

Table 2. Summary of interview questions

Table 3. Summary of documentation
Chapter 1

Introduction

Software is more than just a computer program; in fact it also includes documentation and configuration data, which is needed for the program to operate correctly. A professional software developer is developing software that other engineers also are going to work on and need to understand, a way to transmit this knowledge is by writing software documentation (Sommerville, 2011).

All systems need to have documentation and it is one of the factors that contribute to a systems success (Sommerville, 2011). Forward (2002) defines software documentation as “an artifact whose purpose is to communicate information about the software system to which it belongs” (p. 1).

The role of documentation differs between when it is used and who is using it. In the beginning of the software lifecycle, managers wants it to be able to plan and budget the project, developers want it to know what and how they should build the system. When a system is under maintenance the purpose of the documentation change, developers want it to know what they should change in the system and they want some references to be able to understand how the system is built. Information about systems architecture and functions is especially helpful for new developers that previously have not been involved in the building of the system (Sommerville, 2011).

A system without documentation forces everyone involved to rely on the source code, practically this means that everyone has to rely on the programmers that know the source code.

The role of documentation is not limited to one project, Cockburn (2001) mentions that documentation needs to prepare for the next projects in order be useful in other projects too.

Different organizations handle software documentation differently, some organizations follows traditional software development methods and spend a lot of time on documentation, and more agile organizations never write a single document. According to studies a ratio of 11% of software projects cost are spent on documentation (Sanchez-Rosado, Rodríguez-Soria, Martín-Herrera, Cuadrado-Gallego, Martínez-Herráiz & González, 2009).

Spending a lot of money and time on documentation is highly criticized and a lot of people claim that most documentation is never used. But even if most documentation is never used most software projects do need some kind of documentation in order to succeed with the task undertaken and to transmit knowledge about the systems (Sommerville, 2011).

What needs to be documented is highly debated, traditional software development standards often require up to 50 documents to be considered documented. Newer lightweight documentation approaches is replacing the more traditional and they recommends just enough amount of documentation. But what is just enough differs from project to project (Sommerville, 2011).
1.1. Problem Definition
Software documentation is a way of transmitting knowledge about systems and is essential for a systems success (Sommerville, 2011). This study is going to be conducted at a company that has complex web based systems without any documentation. The systems are essential for the organization and handles everything from orders, production, invoicing, file sharing and employee-client managing.

The lack of documentation has lead to problems, not all employees know how the systems works and new developers have a hard time learning and understanding the systems. By having software documentation this problem could be solved (Sommerville, 2011).

The complexity of the systems has lead to problem with determining what kind of information and details that are relevant to document. By documenting the systems the company hope the knowledge about the systems will be greater and the ROI of creating documentation will be higher than not having any documentation.

1.2. Research Question
The lack of documentation has lead to problems with the sharing of knowledge about the web systems. People do not know how the system works and operates. To solve this, software documentation is needed:

*RQ1*: What information and details are relevant to document in web systems?

1.3. Delimitations and Limitations
This study is going to be limited to identify what kind of information is relevant to document, deciding the level of details and creating of a documentation process.

This study is limited to one case study. The time limit has been 15 ECTS, which is equal to four months study with a 50% workload. The time for data collection has been limited to six weeks in the beginning of the study where the researcher has been located at the office of the company.

1.4. Research Purpose
The purpose of this study is to identify what information and details that are relevant to document and create a cost-efficient software documentation strategy for the case company.

1.5. Expected Contribution From This Study
The expected contribution from this study is to provide the company with a software documentation strategy. The software documentation strategy will include guidelines of what information and details are relevant to document and a process of how they can create documentation.
Chapter 2

Theoretical Background

2.1. Agile Documentation
As a response to traditional documentation standards programmers and consultants went together to create a method that create just minimum amount of documentation. Agile methods advocate that the focus is to write code and working software rather than documentation, which is also stated as one of the four values in the Agile Manifesto:

“Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan”

(Manifesto for Agile Software Development, n.d.)

Agile teams often claims documentation is a waste of time since most documentation is unused (Sommerville, 2011). Sommerville (2011) argues that even though most documentation is unused and out of date it is still necessary for most software development projects to succeed.

Berglund & Priestley (cited in Rico, D. F., PMP, CSM. n.d. p.1) identified 25 guidelines of how to create documentation in an agile way, the seven first are the following:

1. Use traditional software document standards as "guidelines,"
2. Start off small and incrementally build documentation much like the software
3. Keep the documentation to a very minimum,
4. Use electronic or online documentation instead of printed documents,
5. Make it publicly available,
6. Distribute the documentation via the Internet,
7. Allow anyone to add to or update the documentation,

2.2. Types of Documentation
There is a lot of documentation that can be written, but in order to keep it cost efficient it should be minimized. Sommerville (2011) categorize documentation into user documentation
and system documentation. This study is going to focus on deciding what kind of information that is relevant to document, regardless of its type.

Ruping (2003) also mention that different people are interested in different aspects of a system. In order to create just enough amount of documentation you have to be able to state the target reader, if you can not then the there is no point for the documentation (Ruping, 2003). The target reader should be mention in the front of every document and with a short explanation of what knowledge and background the target reader need to have to be able to understand the information. Ruping (2003) also mention that a documentation writer never should expect too much knowledge from the target reader.

According to Ruping (2003) a document producer need to keep the following three questions in mind when creating documentation:

"Who are my target readers?  
What information do my readers need?  
What will my readers be able to understand?" (p. 25)

2.2.1. User Documentation

When creating user documentation it is important for the document producer to know whom the target reader of the document is and cater it differently depending on the reader. He especially mentions the big gap between documentation for end-users and system-administrators. The documentation needs to be structured starting with an overview and leading the reader to more specific details about the system (Sommerville, 2011).

![Figure 1. Different type of user documentation, their scope and target readers (Sommerville, 2011)](image)

The process of creating good user documentation should include the following five steps according to Barker (2003):

1. User analysis: Basic research about the users and what they want.
2. Planning: Start writing your documentation.
3. Draft review: Seek for feedback on the draft that was created during the previous phase.
4. Usability testing: Test the documents usability empirically.

10
5. Editing: Use the feedback from steps 3 and 4 to create a finalized version.

### 2.2.2. System Documentation

System documentation is another category of documentation, it consist of a number of different documents to describe the system itself. According to Sommerville (2011), for a system to be fully understood the documents describing the design, implementation and testing are essential. If there is a lot of time pressure he argues that every software project at least should document the architectural design, system specification and the source code.

System documentation can include the following documents (Sommerville, 2011):

1. Documentation of the requirements
2. System architecture
3. A description of the program for every program in the system
4. Interface and functionality description of every component in the system.
5. Listing of the program source code. This is mostly made automatic and if the code is well written little extra comments should be necessary.
6. Validation of the program and how it relates to the requirements.
7. To describe known problems a system maintenance guide should be used to document how these problems affect the design and also which parts is software and hardware dependent.

Ruping (2003) categorize documentation into 7 category:

Management documents describe the project overall, it’s scope and schedule.

Specification documents which purpose are to define what the software does and its specifications. It consists of everything from data to the user interface. The documents are used for discussion and are also used during the testing.

Design documents give an overall explanation of how the software works, operates and why. It is mainly used for communication inside the development team but also serves as a knowledge base to pass on valuable information about the system to future projects and developers.

Migration documents are used when the system is going to replace an existing system. The documents describe how the new system should take over the existing data and how it should replace the functionality of the old system.

Test documents are for the test phase, this documents may overlap the specification documents as the testing is used against the specifications.

Usage documents are for describing how a system and parts of it can be used. Usage documents are often used for system integration.

Operation documents explain the operation of the system, how it operates, the problems and also how to solve them (Ruping, 2003).
It is important to keep the documentation up to date as the system is changed. A documentation management system could help to record the relationships and dependencies (Sommerville, 2011).

2.3. Amount of Documentation
The amount of documentation that is required for different projects varies a lot depending on a number of variables involved. The complexity, size, source code and type of technologies involved are just a few variables involved. (Ambler, 2002).

Proponents of traditional methods often recommend write a lot of documentation. Agile documentation proponents often define the required amount of documentation to be just enough for what is necessary for the specific project (Sommerville, 2011).

*Right-sizing means that the level of effort applied to write and maintain the documentation plus the value of that written document should have a greater return on investment (ROI) then not having that information readily available (i.e., the effort it would take reconstruct the information and impact of not having that information for current decisions) (Moreira, 2013, June 15)*

In Agile Modeling (n.d.) a comparison of traveling without a suitcase and software documentation is made: “create just enough models and just enough documentation to get by”. Cockburn (2001) claims that documentation should be light but sufficient.

Beck (2008) goes even further and suggest that no documentation should be needed if you have well-written code and Ruping (2003) simply states that: "Too much documentation is as bad as no documentation at all." (p. 1).

![Figure 2. Relationship between usefulness of documentation and amount of documentation (Ruping, 2003)](image)

As seen in the figure there is a certain point in the relationship between usefulness of documentation and the amount of documentation where the usefulness decrease when more documentation is added. Ruping (2003) argues that when there is too much documentation it becomes so hard to find relevant information that the usefulness decreases. By keeping the documentation light more time can be spent on doing high quality documentation and keeping it up to date (Ruping, 2003).
2.4. Creation of Documentation
The figure illustrates a document late strategy for agile documentation. The curves are illustrating the total effort invested in documentation. The figure also shows where different type of documentation can be produced but it should not be seen as exhaustive. The main difference between a traditional approach and agile approach is the amount of documentation but also when it is created. A traditional approach produces a lot of documentation in the beginning of a project and agile approach tends to create documentation as you go and when it is needed (Agile Modeling, n.d.).

![Figure 3. Relationship between total effort and time in documentation late strategy and traditional documentation approach (Agile Modeling, n.d.)](image)

Continuously documentation strategy is a more disciplined agile approach. It means that you write all your documentation continuously as you go. The purpose is that if a system is going to be shipped after an iteration it also has some documentation that can be delivered at the same time (Agile Modeling, n.d.).

![Figure 4. Relationship between total effort and time in continuously documentation strategy and a traditional documentation approach (Agile Modeling, n.d.)](image)
Agile Modeling (n.d.) argues that both agile documentation approaches, documentation late and continuously documentation can be used in an extreme way. One extreme way is to write all documentation at the same time as you are developing the software, this approach has the advantage that it is possible to capture all details without forgetting why different decisions were made. Another extreme approach is to write all documentation when the project is finished. The advantages are that you know what you have built and that it is not likely to change dramatically anytime soon. The disadvantages are that some important people in the project can have left the organization, that the people in the project have forgot about the system but also that there is no motivation to write the documentation. According to Agile Modeling (n.d.) the best documentation approach is to do something in between this extreme approaches by start writing documentation as soon as the project stabilizes.

2.5. Writing Guidelines
The documentation should start on a small scale, it is much more promising that a documentation strategy is going to work if it is done small without trying to achieve everything at one time (Ruping, 2003). Both Ruping (2003) and Brooks (1995) argues that detailed material is only useful for experts, and that information that covers the “big picture” are more useful and can even be understood by people that are not expert in the specific system.

“Most documentation fails in giving too little overview. The trees are described, the bark and leaves are commented, but there is no map of the forest.” (Brooks, 1995, p.165)

The “big picture” should explain the underlying architecture of a systems construction. Ruping (2003) argues that all software projects need to have an overview over the system and that the “big picture” often serves as a bridge between written and face-to-face documentation.

Documentation can only be useful if it is of high quality and Ruping (2003) mention several times the importance of keeping the documentation of high quality: accurate, up-to-date, and highly readable.

To help the reader and to avoid misunderstandings, a document often needs a glossary for definition of words, a guideline for readers, documentation history as well as reference to other documents (Ruping, 2003).

2.6. Documentation Review
Writing good documentation is hard and to minimize misunderstandings a good documentation review culture could improve the quality of the documentation (Ruping, 2003). Giving and taking criticism is hard for some people. For people to accept criticism better, a review should not only focus on the negative aspects but also the positive aspects, the negative aspects should be given together with suggestions of how it can be improved (Coplien, 2000). Ruping (2003) also mentions that by providing positive feedback you also help the writer to know what information is good in case the writer was unsure about whether to replace or keep different parts of the text.
For critical comments to be useful a community of trust need to be formed between the author and reviewer (Coplien, 2000). All documentation need to be reviewed before it is officially distributed to be sure that the documentation is accurate and easy to understand, it is also a way for the reviewer to gain knowledge (Ruping, 2003).

According to Ruping (2003) a documentation review should address five questions:

- Does the document meet its goals, and will it be of use to the readers?
- Is the document technically accurate, and does it provide the right level of technical detail? Is the overall structure and organisation right?
- Does the document provide enough examples to be comprehensible? What about layout and language? (p. 175)

2.7. Kruchten 4+1 Model

The 4+1 model is a way of documenting the architecture of a system in a way that it meets the different stakeholders needs. The model consist of five views, logical, implementation (development), process, deployment (physical) and use case view (Kruchten, 1995). The naming of the views are not concise in the literature.

The logical view is mostly for the end users concern and it illustrates the system at a class level. The implementation or development view is mainly for developers, it illustrates how different components in a system are organized and the relationships between them. The process view is for illustrating the business processes and how messages are sent and received within different systems. Deployment view or physical view is for server administrators or people who need to see how the infrastructure and servers are organized. To help the users understand how the system is supposed to be used the +1 use case view can be used (Kruchten, 1995).

Muchandi (2007) developed the model and mapped it to UML and came up with recommendations of what UML diagrams can be used to illustrate the different view. The figure below illustrates the 4+1 model and the red text state which UML diagrams can be used in each view.

![Figure 5. 4+1 model with mapping to UML (Muchandi, 2007)
Chapter 3

Literature Review

The collection of literature for this study has been conducted both before and during the study when there has been a need to get a deeper understanding of a phenomenon. The search for literature has not been limited to one medium; instead several sources have been used to find relevant literature. The main source for finding literature has been Primo, a search engine by Lulea University of Technology (LTU) library. Google Scholar has been used as a compliment to Primo when Primo did not provide a satisfactory result. The literature has been sorted by age and peer reviewed. Not all literature are very new or peer reviewed because the subject is not very well researched. To determine if an older literature work should be used in this study deeper research has been done to see if the principles presented still hold, if in doubt contact has been taken directly with the authors to ask about their opinion and if they would have changed anything if they published it today. That is why even though not all literature in this research is very new all the principles described still hold. If the articles has not been peer reviewed I have done the judgment myself if it is relevant to use the source in my research.

Traditional software development methods use a comprehensive amount of documentation. There are a few documentation standards to use but most of them are old and strict. MIL-STD-1521 was one of the first documentation standards released 1976 and required 41 documents (U.S. Department of Defense, 1985). MIL-STD-498 was released later and required 47 documents, but it also needed documentation for all the subsystem resulting in hundreds of documents (U.S. Department of Defense, 1994).

MIL-STD-498 was developed to IS/IEC 12207 and required 70 documents for commercial systems (International Organization for Standardization/International Electrotechnical Commission, 1995). The cost to prepare documentation for this standard for a software system of 10,000 lines is estimated to be about 3 million dollars (Rico, 2004).

Documentation is a vital part of developing and using a computer-based system. In some commercial organizations, 20 to 40% of the total development effort goes into the documentation of the new system, recording how the new system is to work and how it was developed. (Ralston & Meek, 1976, p. 23)

The proponents of these methods argue that software need to have lots of documentation, otherwise it will be hard to specify, design and maintain a system (Sommerville, 2011).

De Souza, Anquetil & De Oliveriras made a study in 2005 about the documentation that is essential for software maintenance. The study consisted of two surveys. The first survey were answered by 76 software maintainers in Brazil, they were asked to rate the importance of different types of documentation that were useful to be able to understand a system.

The result from the first survey indicates that software maintenance sees the source code and the source code comments as the most important artifact. The second most important artifact
in the opinion of software maintainers was a logical data model. Other artifact that was rated high was non-technical artifacts such as requirement specification and user manual.

In the second survey they asked software maintainers what kind of documentation they had used when maintaining a system. They obtained 237 answers from Brazilians software maintainers.

The second survey shows that the top ten documentation artifacts that actually are used are the following ten; the percentage numbers indicate the number of maintainers that used the artifact (De Souza et al., 2005):

1. Source code 95.1%
2. Unitary test plan 75.0%
3. Comments 71.4%
4. Requirement description 59.5%
5. System test plan 59.3%
6. Logical data model (MER) 56.3%
7. Component specification 52.2%
8. Physical data model 48.0%
9. Implantation plan 45.1%
10. Acceptance test plan 44.2%

Another study made in 2003 by Lethbridge, Singer & Forward was studying how software engineers use documentation in practice. The study consisted of interviews and web questionnaires responded by software engineers at both corporate and government sites. The key findings included that systems often have too much documentation and that documentation of all types are mostly out of date. Even though the documentation is out of date some of it might still be useful, especially the architecture documentation since it gives the maintainers a historical guidance of the system. The reason why most documentation is out of date was said to be because software engineers only update the documentation that they consider worthwhile to update, even if managers has strict rules on update of documentation. The authors conclude that it is important to create documentation strategy with great relevance that are simple and powerful enough for software engineers to update and that is what this study is going to do (Lethbridge et al., 2003). The costs of creating and updating great amounts of documentation are also significant; a ratio of 11% of software projects cost is spent on documentation alone (Sanchez-Rosado et al., 2009).
Chapter 4

Research Methodology

4.1. Research Method

The purpose of this study is to create a software documentation strategy and an inductive approach has been used. An inductive approach is used when a researcher aim to create a theory from the research as opposite to an deductive approach where the researcher use existing theory to come up with a hypotheses and then tests it empirically (Yin, 2008).

A qualitative research approach is often used for researchers to develop a deeper understanding of a specific phenomenon instead of focusing on statistical analysis as in a quantitative approach. A qualitative approach was chosen to be used in this study. According to Yin (2008) there is a number of research method in the qualitative field of which case study is one method used for investigating a problem in depth in its real life context.

Case study method was considered as the most suitable research method to use in this study, where a strategy should be created and implemented in a real life context. The method has been criticized and Vissak (2010) claims that the result from a case study can be hard to generalize. I have chosen to follow Yin's case study method for this study.

Yin (2008) describes three approaches to do a case study, *explanatory, descriptive and exploratory*. They are all using the same process, which are *design, prepare, collect, analyze and share*.

---

*Figure 6. Case study workflow and how it was implemented in this study*
4.2. Research Design

The case study method can consist of four different types of designs (Yin, 2008):

- **Type 1:** Single case (holistic) design
- **Type 2:** Single-case (embedded) design
- **Type 3:** Multiple-case (holistic) design
- **Type 4:** Multiple case (embedded) design (p. 46)

A single case design is when only one case is studied and multiple-case design when multiple cases are studied. In a holistic case study the researcher has one unit of analysis for each case while in an embedded case study the unit of analysis is split into multiple units (Yin, 2008). In this study type 1, single-case holistic design has been chosen.

In this phase the researcher needs to have an understanding of relevant theory and what has been studied before (Yin, 2008).

4.3. Preparation

In the prepare phase the researcher should prepare for the case study. Prepare for the case study consist of developing necessary skills, getting necessary approvals and search for issues in the research design. According to Yin (2008) the researcher should also become so familiar with the study domain to be able to understand relevant theory and the issues related to it.

The preparation phase in this study consisted of doing a deeper study of existing theory and literature in the field of software documentation and come up with a literature review. As the researcher was located at the company's office the first weeks were used to get an understanding of the organization and the problem. An understanding of the systems and how they are used within the organization were also gained.

During this phase, an agreement of how to handle the confidential information about the company was made as well as an agreement to not reveal the name of the company.

4.4. Collect

In the collect phase the researcher should use multiple sources of evidence, follow a case study protocol and create a case study database. Yin (2008) states that all case studies are “likely to be more convincing and accurate if [they] are based on several different sources of information” (p. 116).

The data was collected during a six-weeks stay at the IT/R&D-department of the company. During the stay an understanding of how the organization works was gathered. It was also possible to see the workflow and the problem in reality and how the solution of this project was going to benefit them.

By staying there for six weeks it was possible to conduct multiple interviews with different people involved at different layers of the organization. It was also possible to go and ask the same people again if follow up questions were needed.
Qualitative interviews were chosen to be use as the primary data collection method. The advantages of using qualitative interview are that the researcher is getting a deeper knowledge about the interview person and his aspects and point of views (Magnusson & Marecek 2015). Both unstructured and semi-structured interviews were conducted. Most of the interviews were unstructured, according to Magnusson and Marecek (2015) unstructured interviews are best suited if a researcher wants to see how the interview person acts in their environment.

After having gathered a wide knowledge of the organizations problems and what information I needed to proceed with my research semi-structured interviews were made. Semi-structured interviews are best suited when a researcher wants to get a deeper knowledge about specific problems (Magnusson & Marecek, 2015).

During the stay participant observations were also conducted. Participant observations are a way to increase the validity of the research and also give the researcher a greater understanding of the context (Dewalt & Dewalt, 2002). The observations were made as focused observation in which the participant guides the researcher of what to observe (Werner & Schoepfle, 1987). Focused observations were chosen to be able to focus on just the specific problem because the participant knew better where the problems was than the researcher. By doing observations it was possible to see how the software development cycle works today and to see how people were trying to tackle the problem with working with a system without any documentation, by doing observation it was also possible to see where the employees think there is a lack of documentation. The people observed were developers and people from the IT service desk.

4.5. Analyze
Yin (2008) describes data analysis as the most difficult part in a case study with qualitative data. The data analysis should be performed parallel with the data collection for the researcher to be able to make adjustments to the research design. This study has been using pattern matching and OTTR for the data analysis. Yin (2008) claims that pattern matching is the technique that is most desirable to use in a case study. Pattern matching is a technique for comparing predicted patterns with the patterns that have been empirically observed and analyzing the gaps between them. The bigger the gap is the easier the matching gets and to analyze the data.

OTTR, observe, think, test and revise is a central concept for data collection and analysis in a case study where observations are conducted. After every observation the researcher should think about the data and information he have collected during the interview. The thinking leads to an understanding of what kind of information he needs to proceed with the research. When the researcher knows how to proceed he goes to the test phase and collect the information he needs. After the test phase the data is analyzed and it might lead to a revision. The process is iterative, the revision might lead to a new test phase and will continue until the researcher is satisfied or when any new data will not lead to new information (GAO, 1990).

4.6. Share
In the share phase the researcher should define the audience and display enough evidence for the audience to come to their own conclusion (Yin, 2008).
The audience for this study is mainly the company that has been studied but also for people interested in the field of software documentation. Since this study has been in a confidential environment and a NDA agreement has been made all evidence and information about the company can only be provided to the company that has been studied. The hidden information should not affect the possibility for readers to make his or hers conclusion of the validity and reliability of this study.
Chapter 5

Data Collection and Analysis

5.1. Methodology for Primary Research

To decide what information is relevant to document and what level of details it should be documented in were done by observing the organization and conducting interviews. The first interviews were done as open interviews to get the respondents personal opinion of the problem and on what kind of documentation is needed, it was also a way for the researcher to be able to understand what kind of information was needed to proceed with this study. The interviews were later made as semi-structured interviews to be able to get into more specific questions. Ruping (2003) and Sommerville (2011) theories about software documentation were used as an inspiration and reference during the interviews. The theory about documentation was discussed with the respondents and their opinions were gathered. By using theory in the interview it was possible for the respondents to get a different view on the problem and raise awareness of other kind of problems that can occur and what can be solved with documentation.

The respondents in this study have all been from the IT/R&D-department since they are the one that is primarily going to use the documentation. The choice of respondents was based on people that are integrating with the system and has a need for documentation, either for reading the documentation or going to produce the documentation. The respondents were also chosen to be from different positions, both to be able to get managers view of the problem, since they are the one responsible for the department and the process but also developers and IT-support people that are going to be affected by the documentation, both for using and producing. Since the data of this study is confidential the respondents has been kept anonymous.

<table>
<thead>
<tr>
<th>Respondent</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Respondent A</td>
<td>IT-manager</td>
</tr>
<tr>
<td>Respondent B</td>
<td>IT-service desk team leader</td>
</tr>
<tr>
<td>Respondent C</td>
<td>Developer</td>
</tr>
<tr>
<td>Respondent D</td>
<td>Server administrator</td>
</tr>
</tbody>
</table>

*Table 1. Summary of respondents*

5.2. Analysis of the Interviews

<table>
<thead>
<tr>
<th>Number</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td>How is the situation today?</td>
</tr>
<tr>
<td>Question 2</td>
<td>How is the process of the weekly updates of the systems?</td>
</tr>
<tr>
<td>Question 3</td>
<td>What is the biggest problem with the lack of documentation today?</td>
</tr>
<tr>
<td>Question 4</td>
<td>How much time are you willing to spend on documentation?</td>
</tr>
<tr>
<td>Question 5</td>
<td>What kind of documentation do you see as important?</td>
</tr>
</tbody>
</table>

*Table 2. Summary of interview questions*
**Question 1. How is the situation today?**

The company has a number of different web systems that handle everything from order creation to production to invoicing. All employees and customers use the systems but they have access to different parts. There is a customer service department for handling questions from the customers and a 2nd line IT-service desk department that takes care of more technical issues as well as supporting internal questions from the organization about the systems.

They do not follow any specific software development methodology. Their development process depends on the size of the project. Most time is spent on updating and maintaining the existing systems; every 2<sup>nd</sup>-week there is an update of the systems.

There is today a ticketing system and an online knowledge base where users can ask questions about how to do different tasks in the systems. The knowledge base is not widely use.

The documentation available is source code comments and some parts of the code are documented with phpDoc. There is also a requirement specification used when doing updates in the systems. Known problems about the system and if they are software or hardware dependent are documented in the service desk portal.

This question was asked to the IT-manager to get an understanding of how the situation is today.

**Question 2. How is the process of the weekly updates of the systems?**

When there is a request of a change in the system a basic mockup is usually used to communicate with the requester. When the requester is satisfied with the mockup a basic requirement specificat is written. The requirement specification together with the mockup is handed to a developer that realize the change.

For smaller changes usually one developer writes all the code himself. When there is a bigger project they usually set a requirement specification that only allows minor changes, then split the project into different phases where they after each phase launch the update.

The work that has been made goes into a 2<sup>nd</sup>-weekly system update. All updates are tested before a tester releases them. There is no test documentation at all which has resulted in that some bugs are launched in the updates.

All respondents agreed that the requirement specification that is used today is satisfying.
Question 3. What is the biggest problem with the lack of documentation today?

The biggest problem is that the web systems are so complex that it is hard to remember what the different systems do and how they work. The company also needs to hire new people who are going to work with the systems, both people at the IT-service desk and developers. There is a need to be able to introduce them to the systems in a way that is efficient and easily understandable and software documentation would be one way to achieve this. The documentation could also be useful for the experienced IT-service desk people and developers if they want to investigate something. When asked directly if there is a need for documentation for less technical people the IT manager agreed and claimed that if the documentation is written in a less technical way it could also be used by other departments to understand the basics of the systems.

Question 4. How much time are you willing to spend on documentation?

The advantages and disadvantages of a traditional contra an agile documentation approach were presented to the IT-manager. An agreement was made that an agile approach would be more suitable to be able to keep the documentation strategy cost efficient and with a low TCO and high ROI.

Further discussion lead to the conclusion that it is more promising to start the documentation process on a small scale and in the future if the documentation process works well and there is a need and motivation for more it could be added later on.

All respondents agreed that the biggest problem is that there is no reference manual and that it is essential that one is created. The reference manual is necessary for the IT service desk and the developers. The IT service desk needs to be able to get an understanding of how the functions work since they receive the most technical questions from the customer service. The IT-manager also sees a reference manual as a way for new employees at the IT/R&D department to get familiar and learn the systems. Additionally, the IT-manager also claims that if it is written in a way that is not very technical the manual could also be used in other departments. With further discussions about Sommerville (2011) claims that different users want different information an agreement was made that the reference manual for the IT department has the highest priority and that an end-user manual that can be used by other departments has a lower priority but is needed.

Question 5. What kind of documentation do you see as important?

All respondents agreed that a manual that in plain text explains the functions of the systems is essential. How this should be done and on which grade of formality it should be written did no one had any suggestions about.

By presenting to the IT-manager the theories of Sommerville (2011) and Ruping (2003) about what software documentation that is necessary, it became clear that there is a need to document the overall architecture and that it is necessary to have a database scheme. The reason why there is no database scheme is because the database does not consist of any foreign keys which makes it impossible to auto generate a diagram, which leads to the
conclusion that the database need to add foreign keys before this diagram will be available. Documenting the techniques used, MVC and ORM, and why they are used was not of interest since they are considered industry standard.

By presenting to the system administrator the user documentation theory of Sommerville (2011) about the need of having an installation document he agreed with the claim. There is today only one person who knows how to install the systems from scratch.

5.3. Analysis of the Observations

The observations were made as participant observation and were conducted to be able to see how the systems work in practice.

The first observation was made of the entire organization; an understanding was gathered of how the organization works, over the organization departments and what they do.

The second observation was made with the IT-service desk team leader and was conducted as a guide through the systems to learn how the information and data flows between the different systems. An explanation was given about how different users interact with the systems and how each page can display different information depending on a number of variables. How and why the systems work differently depending on variables need to be documented.

The third observation was made with a developer to see how the code is documented, organized and mapped to the systems to get an understanding of how it was and how it could be documented.

5.4. Documentation Needed

After analyzing the need of documentation it became clear that the biggest need for documentation is for five different roles, developers, system administrators, IT-service desk, testers and end users. Documentation writer is another role that also has a need for documentation because they need some guidelines of how to write the documentation. The existing systems are very complex and due the limited time for documentation the documentation creating need to be prioritized. The workflow is also going to be changed which means that new procedures is going to need to be implemented in the organization. The agile late documentation strategy is the most similar approach to document the existing systems. The only difference is that there is not going to be any requirement specification over the systems, only over the new function and changes that are going to be made within the systems. A requirement specification over the entire system is nothing that is going to be needed because the system already exist and it is better to document what already exist in terms of description of functions instead.
5.4.1. Documentation for Developers

Requirement specification

The requirement specification is essential for the developers to know what they should build. Sommerville (2011) states that it is one of the most important documents during a software development project. The requirement specification is going to be used to specify the updates of the systems.

According to my interviews the current requirement specification is satisfied why there is no need to make any dramatic changes to it. Ruping (2003) claims that the requirement specification may overlap the test documentation since the test phase is used for testing the requirements. Because there is no current test documentation and the interviews confirmed that there is a need for some kind of test documentation an approach where the requirement specification is going to be used as test documentation for functional requirements was chosen. In the requirement specification there will be a checkbox where the tester checks and sign when a test has been made and if it passed or failed. This is going to help keep the TCO lower and make it simpler for everyone involved to use the documentation.

Process view

(UML Activity diagram)

The process view is for illustrating the behavior of the running systems, the processes and how they communicate. It is a part of the Kruchten 4+1 model and can be drawn with an UML activity diagram (Muchandi, 2007). The activity diagram should illustrate the main business processes, in this case it is how an order is processed in the various systems. This diagram can help developers, and others to get an understanding of how the order is processed and also help the developers to understand where in the systems they are going to make changes and how it will affect the overall process.

<table>
<thead>
<tr>
<th>Developers</th>
<th>Server Administrator</th>
<th>IT-service desk</th>
<th>Testers</th>
<th>End Users</th>
<th>Documentation Writers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement specification</td>
<td>Physical view</td>
<td>Reference manual</td>
<td>Test documentation (In requirement specification)</td>
<td>Introductory manual</td>
<td>Documentation guidelines</td>
</tr>
<tr>
<td>Process view</td>
<td>Installation instructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development view</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database scheme</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Summary of documentation
Development view
(UML Component diagram)

The development view is a view in Kruchten's 4+1 model for illustrating the systems architecture. An UML component diagram can be used for this to illustrate how different components in the systems makes up the entire system (Muchandi, 2007). It should be drawn in a high level only to give the reader an overall understanding of the relations between the different systems and their components. By drawing it in high level it also opens up the possibility to be useful for not only the developers but also others and can be used by others to get an overall explanation of the systems.

Database Scheme

A database scheme has been ranked as one of the most used documentation artifacts in previous studies and my interviews also confirm that it would be helpful for the developers (De Souza et al., 2005). The database does not consist of any foreign keys why it is impossible to automatically create an EER-diagram with relationships between the tables. The only solution for this is to first add foreign keys to the database.

Code documentation

The code documentation is currently already in place. Source code comments are written and phpDoc is used for source code listing. Sommerville (2011) also mentions if the code is well written not much comment is necessary, a listing of the source code can still be useful, for example an listing of the functionality and methods and it can be solved with phpDoc.

5.4.2. Documentation for Server Administrator

Physical view
(UML Deployment diagram)

The physical view is from Kruchten's 4+1 model and can be illustrated using an UML deployment diagram (Muchandi, 2007). This view is to help the server administrator get an overview of the hardware components and their relationship with each other. This should be drawn in a high level only.

Installation Instructions

Sommerville (2011) claims that one kind of user documentation is the installation documentation for the server administrator. This document should explain how to install the systems from scratch. The interviews confirmed that there is a need to document the installation procedure since only one person knows how to install the systems from scratch.

This document is going to include a detail description of how to set up webservers and install the systems. Since the company is working with Gitprojects it also need to include
instructions for how to deploy these projects and all other commands that need to be run in order to install the systems.

To verify that the installation document works the systems should be taken down and installed from scratch.

5.4.3. Documentation for IT-service desk

Reference manual

According to Sommerville (2011) user documentation for experienced users is needed. The document should include all details of facilities in the system and the functionality. All respondents in my interviews confirmed that this document is the most important in the case.

The document should be divided into sections where each section belongs to a system. The section should consist of a brief description and the purpose of the system. Below the description all functionality of the system should be documented in a brief way.

By doing this the document can also be served as a document for new developers to learn from and use as a reference manual when they encounter any difficulties in the system. This documentation will also be useful for the people at the IT-service desk that are going to receive questions about the systems, the document could also help users from other departments that feel that they have the knowledge to understand the content.

5.4.4. Documentation for Testers

Test documentation (Included in the requirement specification)

Test document is one of the essential documents according to Sommerville (2011). Ruping (2003) also mention the importance of validating a product. Ruping (2003) claims that the test document may overlap the requirement specification since the testing is against the requirements. The interviews confirm that the lack of test documentation today is causing bugs in their releases.

To save time and do the testing more efficient the functional tests are going to be combined with the requirement specification. There is going to be an extra checkbox and signature box where the tester signs when a test is done and if it passed or failed.

The problem with this strategy is going to be that only functional tests are going to be tested. Non-functional tests are not going to be tested which may result in that such errors are going to be discovered later. To solve this a strategy need to be created of how to test non-functional tests.
5.4.5. Documentation for End Users

Introductory manual

An introductory manual for the end users that are written in a very simply way to understand the overall system and how to use it is needed. As Sommerville (2011) mention there is a big gap between system administrators and end-users, they are interested in different aspects and level of details of the systems. This is also confirmed by my interviews but has a lower priority in the organization.

5.4.6. Documentation for Documentation Writers

Documentation Guidelines

This document is not described in any of the software documentation theory. But to work with knowledge management and keep more information in the organization this document is going to serve as a guideline for the documentation writers to know how they should write the documentation. It is going to give some brief advices and what they should think of when they are writing documentation.

5.5. Documentation of Updates Workflow

The figure below shows how the workflow needs to be changed and implemented in order for this documentation strategy to be efficient. Normally there are just minor updates which will not affect the architecture.

1. A decision is taken about what need to be updated in the systems.
2. The owner of the requirement writes a requirement specification from the template.
3. The developer takes the requirement specification and implements the update.
4. During the coding the developer writes source code comments and comments with phpDocs.
5. The developer hands what he has built to a tester that test the function. The tester is using the requirement specification and tests the functional requirement. He writes in the requirement specification if the test passed or failed. If it failed he write a comment about what failed and gives it back to the developer.
6. When a test has been passed the documentation writer update the reference manual and the user manual.
7. When it is documented he asks a colleague to review and approve all documentation he has updated.
8. The update is released.
9. A notification is sent to the users that the system has been updated.
5.6. Process Standards

All documentation should be stored and updated online on Google Drive where all internal users have the permission to read and edit the documents. This way it is clear for everyone that the latest version of the documents always is on Google Drive. Some documents, for example the reference manual can be used as a paper copy but it should be clear for the reader that the paper version might not be the latest version. That is the reason why every document should have a version number and a documentation history. Discussions have been held about using another tool such as Confluence for the documentation. Google Drive is a tool that is currently used within the organization for other kind of documents and is satisfying why it is the most suitable choice to continue using it.

All documents that have been created or updated should go through a documentation review. The documentation review should be performed as soon as possible after the text has been
changed to minimize any misunderstandings. It is the reviewers responsible to approve or come with feedback on what should be edited, if the reviewer and writer cannot agree ask a third part.

The layout of the documents should follow the organizations graphical profile and each section of the documents should have a brief description about the section.
Chapter 6

Conclusions

6.1. Overview
This research used a qualitative approach with interviews and participant observations to be able to understand the organization and their need for documentation. When an understanding of the need was gathered it was mapped with the existing theory in the field to come to a conclusion of what documentation is the most necessary.

During the study of theory there was a clear line between old literature and newer literature. Older literature about traditional software development approaches argued that extensive documentation was needed and newer literature about agile documentation argued that most documentation is unused and that it is better to minimize the amount of documentation to keep the ROI higher.

Previous studies has been made about what kind of documentation software maintainers most often use which shows that the source code and the database diagram is the most used documentation and that they do not really trust that external documentation are up to date (Lethbridge et al., 2003). This states the importance of keeping the documentation up to date. By not keeping the documentation up to date the value of it will decrease over time until it reach a threshold where the documentation is no longer useful and the source code will again be the only documentation. Ruping (2003) also argues that too much documentation is as useful as no documentation so there need to be a balance of the amount of documentation and to always keep it up to date.

The result in this study shows that the organization need to document various information, it would be easy to add more documentation later if there is need for it in the future but the documentation strategy presented in this study should cover the need for most people but still in way that the amount is reasonable to document.

This study has also focused on the cost perspective of creating documentation, more specific the ROI of software documentation. It is important to remember that the ROI will differ a lot depending on the organization. A very big organization with hundreds of thousands employees and millions of customer will have more resources than small organization to create a greater amount of documentation. A larger organization will also have a more people that can use the documentation, which leads to a higher ROI.

The result from this study is mainly interesting and usable for the organization being studied. The result can be used as a reference for other organizations with a similar problem but it is important to remember that every system has its own needs.

6.2. Conclusion from this Study
The conclusion from this study is that it takes time to document an existing system that currently has no documentation. There is a need to specify whom the documentation is for
and the purpose, it was also clear that different readers want to have different kind of information. It is also important to determine how much time and money an organization wants to spend on documentation, in this case as little as possible, which is why an agile documentation approach was chosen.

Due to the fairly large size of the company with over 500 employees the ROI will likely be greater than not having any documentation. Especially the introductory manual aimed for the customers due to a large customer base. If the documentation is kept up to date the ROI will be even higher in the future if the organization grow in number of employees, which can gain benefits from the documentation.

What was learnt is that documentation that covers the big picture is what is the most needed. To document the high level architecture Kruchten’s 4+1 view model can be used, in this study we have excluded the usage of the logical and the +1 view because there simply were no need for it. To document the other views an UML component diagram, deployment diagram and an activity diagram over the most important process in the system can be used. For more detailed information the developers most often use the source code. Documenting the micro management would be way too costly and most likely not very useful, it would also be very hard to keep the documentation up to date.

A reference manual was what was most requested in this organization, there were a few different approaches available to create one. What was chosen was a manual with a brief description of the systems and a brief description of all the functions in the systems. An approach where the documentation had been mapped to the code and database could have been used but was decided not to since the purpose was just to give an overview and according to previous studies the software maintainers wants to use the source code as their primary tool to research problems (De Souza et al., 2005). A user manual aimed for the end-users is also needed.

The existing requirement specification for the updates of the systems is satisfying and should still be used but also work as test documentation for functional requirements. By doing this it simplifies the work for the developers and there is only one document to use and update.

If the there is time and willingness to take the documentation even further a development of a better documentation in the test phase should be considered. A proper test plan and test protocol would be a way to minimize the number of bugs in the releases and to be able to follow up the testing in a better way.

Implementation of the strategy should be started in a small scale; it is way more promising and will minimize the risk for lack of motivation to keep it up to date.
Chapter 7

Reflections and Suggestions for Future Research

What was learnt from this study is that documentation is an abstract field, there is no right or wrong, all systems and organizations has their own need and it is up to every organization to decide whether how much time and money they want to spend on documentation. This study has been conducted because putting documentation at the side has lead to problems in this organization. What has been presented in this study is what information is most relevant to document to solve this problem.

It was learnt that the documentation that is most needed is information that captures the big picture of the systems and not specific details. Previous studies also confirm that developers preferably use the source code as the first choice of documentation (De Souza et al., 2005).

The research area of software documentation is not very mature, only a few relevant studies have been found and most of the ones that have been found have been quite old. Most studies found have focused on documentation during the development of a new system. This time when the agile methods are growing in popularity and documentation is put at the side there is a need for further research about the costs and ROI of documenting an existing system that currently has no documentation. A method to calculate this costs based on variables such as the organizations resources, number of employees and how many that are going to use the documentation would be very helpful for organization to determine weather they should invest in documentation or not. There is also a need for research about the possibility to automatically create documentation of the architecture and to create an artifact that solves this if it is possible.
References


Appendix

Documentation Summary & Guidelines

Overview Description
The purpose of this document is to give an overview of the documentation that exists over the systems. It also consists of a guideline of how to write documentation.

The following is a list of documentation available over the systems:

**Requirement specification**
Target reader: Developers
Includes: Requirements

**Physical view**
Target reader: Server administrator
Includes: UML Activity diagram

**Process view**
Target reader: IT-service desk
Includes: UML deployment diagram

**Development view**
Target reader: Developers
Includes: UML Component diagram

**Database scheme - Missing today**
Target reader: Developers
Includes: EER-diagram

**Installation document**
Target reader: Server administrator
Includes: How to setup web server, deploy git project with the portal, other commands that need to be ran

**Portal Reference manual**
Target reader: IT service desk
Includes: Description of systems and functions

**Test documentation (Included in requirement specification)**
Target reader: Testers
Includes: Functional test pass/fail and test comments
Code documentation
Target reader: Developers
Includes: Source code comments and phpdoc

Different people are interested in different aspects of the system. It is important for the document producer to know who the target reader of the document is and cater it differently depending on the reader. For example something that is written for system administrator will not make any sense to an end-user.

If you can’t state the target reader then there is no need for the documentation. The target reader should be mention in the front of every document to make it clear for the reader if he is the target reader of the document. Also write a short explanation of what background and knowledge the reader needs to be able to understand the document. Never expect too much knowledge from the readers.

When writing documentation, keep the following three questions in mind (Ruping, 2003, p. 25):
“Who are my target readers?
What information do my readers need?
What will my readers be able to understand?”

The documentation needs to be structured starting with an overview and leading the reader to more specific details about the system.

“Most documentation fails in giving too little overview. The trees are described, the bark and leaves are commented, but there is no map of the forest.” (Brooks, 1995, p. 125)

Write just enough amount of text, light but sufficient. No one wants a 2000-pages document. Shorter is also easier to keep accurate and up-to-date.

Keep a documentation history where you write who made the change, when and what changes has been made to a document. A document also often needs a glossary for definition of words, a guideline for readers and references to other documents.

When a document is finished or edited, always let a colleague review the documentation to make sure that everything is clear to avoid misunderstandings. It is also a way for the colleague to receive knowledge about the update. The review should not only focus on the negative aspects, it should also focus on the positive. For a review to be efficient a community of trust must be formed between the author and reviewer.

The documentation review should address the following five questions (Ruping, 2003, p. 175):
“Does the document meet its goals, and will it be of use to the readers?
Is the document technically accurate, and does it provide the right level of technical detail? Is the overall structure and organisation right?
Does the document provide enough examples to be comprehensible? What about layout and language?”

All documents should be stored on Google Docs. This way the documents can be accessed easily by everyone and it’s easy for people to search in the documents. By having the documents on Google Docs the reader knows that he is reading the latest version of the document.
Remember, documentation can only be useful if it’s of high quality. Write accurate, keep the documentation up-to-date, write concise and well structured document.