Université d'Ottawa • University of Ottawa

Software Documentation – Building and Maintaining Artefacts of Communication

By

Andrew Forward

Thesis

Presented to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree Master in Computer Science

Ottawa-Carleton Institute for Computer Science
University of Ottawa
Ottawa, Ontario, K1N 6N5
Canada

ACKNOWLEDGEMENTS

A very special, and well-deserved, thank you to the following:

a) Dr. Timothy C. Lethbridge. As one of my undergraduate professors, he first introduced me to the concept of Software Engineering. As my advisor, he helped me to successfully coordinate my ideas, offered direction with the implementation of those ideas, provided insight in the interpretation of my work, and supported discussions of all the above. Finally as a friend and mentor, giving rich critiques of my research, methods and writings; he helped to improve all three.

b) The Knowledge Based Reverse Engineering (KBRE) group including Abdou, Wahab, Sergei, Souhail, Edna and Adam. Our weekly meetings, the lab and informal discussions about software engineering, reverse engineering and documentation were of great help.

c) My family and friends. Thank you and much love to my mom (and editor) Jayne, my father Bill, and my sister Allison. And of course my close friends; especially Ayana, Aram, Clifton, Hani, Jeremy, Mike and Travis for listening when at times I would ramble on about school, my thesis and research. Richard Seniuk and Charles Gruchy also helped in the final editing process, merci.

d) Software professionals around the world. Sincere thanks to the individuals that participated in my research, published valuable references for my writing, as well as to those in the various news-groups about software engineering that I follow. Your knowledge and insight helped provide the necessary substance for my work.

ABSTRACT

Software documentation is an important aspect of both software projects and software engineering in general. In fact, documentation engineering has become a popular sub-domain in the software engineering community. Unfortunately, the current perception of documentation is that it is outdated, irrelevant and incomplete. For the most part, this perception is probably true. Regrettably, the documentation concern cannot be resolved by simply mandating more and better documentation. This approach fails to resolve the problem as the solution ignores the fundamental goals of software engineering. The role of documentation in a software engineering environment is to communicate information to its audience and instil knowledge of the system it describes. Documentation should efficiently allow for future software development without hindering current progress.

An engineered solution to the documentation problem would involve allocating appropriate resources to document adequate knowledge about the system to the extent that both current and future development will optimally benefit. Unfortunately, neither do we fully understand the impact of documentation on current or future development, nor what aspects of documentation contribute to its ability to communicate effectively. We do not really know to what extent we should document in order to balance the trade offs between, on the one hand, allocating too many resources for documentation thus hindering present development; and, on the other hand, not allocating enough resources and thus hindering future development. To complicate matters, increasing documentation resources does not necessarily improve future software development (the ultimate goal of documentation) because we do not really understand what defines documentation quality.

Our research focuses on the issue of documentation quality. In particular, which attributes of documentation make it effective to the audience and how can this information be monitored and parameterized to provide a better perspective about the relevance of documentation in a software project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

CHAPTER 1: INTRODUCTION

Software documentation can be defined as,

> *an artefact whose purpose is to communicate information about the software system to which it belongs.*

Its role in a software project is not well understood. Many software professionals profess to the importance of documentation in a software environment without a strong grasp of what constitutes quality documentation. In fact, documentation in practice is frequently incomplete and rarely maintained and hence is often believed to be of low quality. Yet, software continues to be designed, constructed and delivered. The documentation problem is usually described as either a lack thereof leading to incomprehensible code; or as something unhealthy that hinders the understanding of the system on a whole because the right documents and appropriate information are difficult (if not impossible) to locate and properly utilize.

The seemingly common-sense approach to improving software documentation – that is, properly documenting all aspects of a software system while maintaining its integrity as changes occur – may not apply, at least in its entirely, to software engineering. At face value, up-to-date, accurate and complete documentation is preferred over outdated, inconsistent and incomplete documentation. However, in the realm of a real software environment, where other factors beyond documentation must be taken into consideration, maintaining documentation may not be the most appropriate or beneficial approach to software development.

Our research will investigate the role of software documentation. Based on our findings, we will introduce a document maintenance model to help cope with the complications arising from the use and maintenance of software documentation.

1.1 Antecedents

Since most of a Software Engineer's time will be spent doing maintenance [52], it seems appropriate that software documentation should be an important aspect of the software process.

But, what constitutes good documentation? Most individuals believe two main requirements for good documentation are that it is complete and up-to-date. We hypothesise that other factors may have a greater impact on documentation relevance than has been previously thought.

We must also consider the issue of the applicability and usefulness of a document. Can complete and up-to-date documents that are rarely referenced or used be considered of high quality? What consideration should be given to incomplete, highly referenced documents that appropriately communicate to their intended audience?

To gauge the quality of documentation, factors beyond its completeness and being up to date should at least be considered when discussing documentation relevance.

Our work is driven to uncover factors, preferably measurable ones, that contribute to (or hinder) documentation relevance. We hope to exploit the effects of these factors to better predict the effectiveness of a document based on the current environment where that document exists.

1.2 Motivation and Objectives

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

A second observation was that several small to medium-scale software projects had little to no software documentation. Individuals in these software teams said they believed in the importance of documentation, but timing and budget constraints left few resources to fully (and sometimes even partially) document their work.

The primary questions arising from the above interactions include:

- How is documentation used in the context of a software project? By whom?
- How does that set of documents favourably contribute to the software project (such as improving program comprehension)?
- How can technology improve the use, usefulness and proper maintenance of such documentation?

Team members of one of the large-scale projects sought practical solutions to organizing and maintaining documentation knowledge. Meanwhile, the individuals in the smaller projects were looking for the benefits of documentation from both a value-added and a maintenance perspective.

Based on the observations above, our research objectives are summarized as follows:

- Uncovering current practices and perceptions of software documentation based on the knowledge and expertise of software professionals.
- Exploiting the information gathered from software professionals to help create suitable models to better define and gauge documentation relevance.
- Implementing an appropriate software tool to put into practice the models uncovered above to better visualize documentation from a relevance perspective.
- Validating our documentation models by experimenting with our approach under several software project parameters.

1.3 Audience

Our research presents important findings for various reasons and to several audiences:

- Individuals interested in documentation technologies can use the research to better understand which existing technologies may be more appropriate than others, and why.
- Software decision makers can use the data to justify the use and selection of certain documentation technologies to best serve the information needs of the team.
- Using the results of our work, better documentation tools and technologies might emerge. Software designers will have a better grasp of the role of documentation in a software environment and under what circumstances it is, or can be, used.
- Our proposed models of documentation relevance provide a framework from which more feature-rich applications can be explored to better support the array of functionality offered by our models.

To a smaller extent, our work also contributes a body of knowledge relating to other facets of documentation including the current use and perception of software documentation tools and technologies. This information could be useful to individuals involved in documentation engineering, or that are involved in the documentation process.

1.4 Organization

The thesis begins with a background of the related work and literature that was referenced throughout our research. Following this, we introduce our empirical study of software documentation, followed by an analysis of the results. This analysis forms the basis and justification of our work. We analyzed the data from a document relevance perspective, looking at how documentation is presently used as well as how its attributes and external factors affect its usefulness. We then analyzed the results from a document engineering perspective, looking at how technology is both currently used and potentially could be used in the software documentation domain.

Following our analysis of current software documentation practices, we will introduce the concept of document Aura as

> *a prediction of an artefact's relevance in a particular situation based on an artefact's usefulness, referential decay and authority (hence, Aura). A measure of Aura is a prediction based on the environment in which an artefact exists as well the conditions under which the prediction is requested.*

Drawing from other computer science domains that concern profiling, ranking and relating information, as well as using basic heuristics about information and documentation, we will describe a model to track and gauge documentation maintenance. We will describe the software architecture and implementation of our model. After which, we will evaluate the effectiveness of our ideas by studying how developers interact with our model (implemented as a software documentation tool) during software maintenance.

For more information about the vocabulary used throughout this document, the following two options are available. First, a glossary is available at the end of this document. Second, most glossary terms are defined within the document, on its first use; such as document Aura above.

In detail, the chapters have been organized as followed.

Chapter 2 reviews the available literature about software documentation and introduces the background essential for the remainder of our work.

Chapter 3 discusses the survey we performed in April 2002. The chapter also provides preliminary results and analysis based on the survey data.

Chapter 4 elaborates on the findings presenting in Chapter 3. In particular, this chapter focuses on the current roles and perception of documentation. This chapter presents a hypothesis suggesting low correlation between documentation quality and documentation maintenance.

Chapter 5 analyses the survey results from a documentation engineering perspective. The chapter highlights the current use of documentation technologies and the participants' preferences (and aversions) towards those technologies. This chapter introduces the requirements of a software solution that addresses several documentation issues described in previous chapters.

Chapter 6 investigates an approach to modeling documentation relevance; known as the Aura of a document. The metrics we have applied to predict relevance are based on the analysis of related literature, including heuristics of knowledge ranking and coordination, and our findings from the survey we conducted on documentation.

Chapter 7 describes the software implementation of a system that tracks document Aura. This tool aims to manage project artefacts to better predict their future need (as both a source of information during searching and browsing and tasks related to document maintenance). The software is the culmination of the research presented in previous chapters.

Chapter 8 provides empirical evidence suggesting the value of tracking document Aura. Our experiment involved novice and expert users of a small VCR software simulator system.

Chapter 9 presents conclusions, future research, and summarizes to what extent we achieved our objectives.

CHAPTER 2: BACKGROUND

The literature we consulted throughout our research has been focused on the following areas of study:

- Software Documentation. Obviously we are concerned with past research that helps define the role of documentation in a software environment as well as provides an analysis of the contribution of documentation to a software project.
- Software Engineering Principles and Heuristics. Looking beyond documentation engineering, we were interested in integrating proven software engineering practices into the realm of documentation.
- Information metrics. This section analyzes how information is processed, ranked and prioritized. Much of the work in this section has typically been applied to fields outside that software documentation, but may in-fact be quite useful.

2.1 Defining Software Documentation

This section investigates the perceived purpose of documentation in a software environment. First we investigate the idea that documentation is more than textural descriptions of a software system, but more importantly, a form of communication among team members. Next, we compare the idea of models versus documentation. After which, we analyse the role of documentation in a software project and finally we discuss the perceived quality of documentation based on its being up-to-date.

Documentation standards proposed by the IEEE (including 829, 1074) and ISO (such as 12207, 6592, etc) help define the contents of software documentation under certain situations. Our approach seeks to define documentation at a much more abstract level. Although it is important to understand and define what the contents of software documentation should be (as done by the standards listed above), we must first understand the goals of software documentation as a whole.

2.1.1 Documentation is communication

Scott Ambler [1] describes the critical components of software documentation from an agile perspective. We believe Ambler's descriptions of documentation are suitable for most software projects, agile or not.

The key elements in Ambler's list of recommended documentation traits describe a fundamental issue of documentation being communication, not documentation [1].

Andrew Hunt and David Thomas [26] explain that communication is achieved only when information is being conveyed. Based on the assumption that documentation is communication, the goal of a particular software document is to convey information. This information may not necessarily be completely accurate or consistent.

Conversely, Brian Baurer and David Parnas [7] argue that precise and mathematical documentation improves software maintenance. They believe that it is to everyone's advantage to invest in preparing and maintaining such documentation. Although Baurer and Parnas' results did agree with their original hypothesis that mathematical documentation improves maintenance, we do not believe the efforts to produce and maintain precise mathematical models of a software system are feasible for software professionals in the industry as a whole. We do not disagree with the premise of the results, which is, that mathematical documentation improves maintenance. Instead we are concerned with the applicability of their approach to documentation. The paper reports on a project with only 500 lines of code and the techniques were applied in an educational environment. Several real-world constraints were not applicable to their case study and, as such, the results are probably ill-suited for documentation in practice.

Ned Chapin [14] states that the value of software can be determined by many factors. One of the four main issues related to software value is its documentation. Chapin describes several main concerns regarding documentation: quality, obsolescence or missing content. To preserve and enhance software value, Chapin recommends to "keep the documentation current and trustworthy" [14]. It is important to note that merely being current and trustworthy do not necessarily imply applicable or useful. In a separate paper [13], Chapin states that as content becomes obsolete, confidence decreases and the cost of software maintenance increases.

Ryszard Janicki, David Parnas and Jeffery Zucker [27] describe the use of relations (represented as tables) as a means of documenting the requirements and behaviour of software. Their work highlights two key limitations to current documentation practices. Janicki et al state that documentation continues to be inadequate due to the vagueness and imprecision of natural languages. These individuals believe that even the best software documentation is unclear. Janicki et al suggest that the limitation in natural languages is a primary cause of informal documentation. They further add that this informality of the documentation in turn cannot be systematically analysed, resulting in inconsistent and incomplete information. Janicki et al then present a solution to building consistent and complete documentation by describing software using tabular mathematical relationships. Although we agree with the motivation for their work, their claim that documentation must be complete and consistent can be somewhat misleading as it ignores what we believe to be the fundamental issues of documentation; communication. Instead, we accept that by understanding the limitations of our ability to communicate effectively, we are better able to deliver artefacts that serve a communication need.

Alistair Cockburn, author of Agile Software Development [15], further describes the limitations of our ability to communicate by stating,

> "We don't notice what is in front of us, and we don't have adequate names for what we do notice. But it gets worse: When we go to communicate we don't even know exactly what it is we mean to communicate. The only thing that might be worse is if we couldn't actually communicate our message." [15] (page 7-8)

Cockburn continues by saying that managing the incompleteness of communications is the key to software development. In regard to software documentation, Cockburn believes that documentation should be sufficient to the purposes of the intended audience.

Cockburn compares software development to a game whose two primary goals are to "deliver the software and to create an advantageous position for the next game, which is either altering or replacing the system or creating a neighboring system" [15] (page 36). Cockburn also describes intermediate work products as any artefact products that are not directly related to the primary goals of the software game. Cockburn then describes the importance of intermediate work products as follows.

> "Intermediate work products are not important as models of reality, nor do they have intrinsic value. They have value only as they help the team make a move in the game. Thus, there is no idea to measure intermediate work products for completeness or perfection. An intermediate work product is to be measured for sufficiency: Is it sufficient to remind or inspire the involved group?" [15] (page 34)"

Alistair Cockburn makes an important comment about intermediate work. First, it is our assertion that most software artefacts apart from the system itself represent intermediate work, including software documentation. Although we believe these artefacts are important to the success of the software game, they are not directly related to the prime goals of software; that is, delivering software. Hence, using metrics based on completeness and being up-to-date are inadequate at reliably ranking documentation effectiveness. Instead, metrics that reflect sufficiency would provide a more appropriate technique to document ranking.

Scott Ambler delivers a similar message in his writings about agile documentation [1]. We believe the strength of documentation lies in its capacity to deliver knowledge and fulfil a communication need and not necessarily to reflect the most current, complete or accurate reflection of the system it describes. As a consequence, metrics that provide insight about, or attempt to predict, a document's ability to illicit knowledge in its readers would contribute to improved documentation ranking techniques.

2.1.2 Software Models Versus Documentation

Scott Ambler describes a software document as "any artifact external to source code whose purpose is to convey information in a persistent manner" [1], and a software model as "an abstraction that describes one or more aspects of a problem or a potential solution addressing a problem" [1].

Ambler describes the relationship between models, documents and documentation below in Figure 2-1.

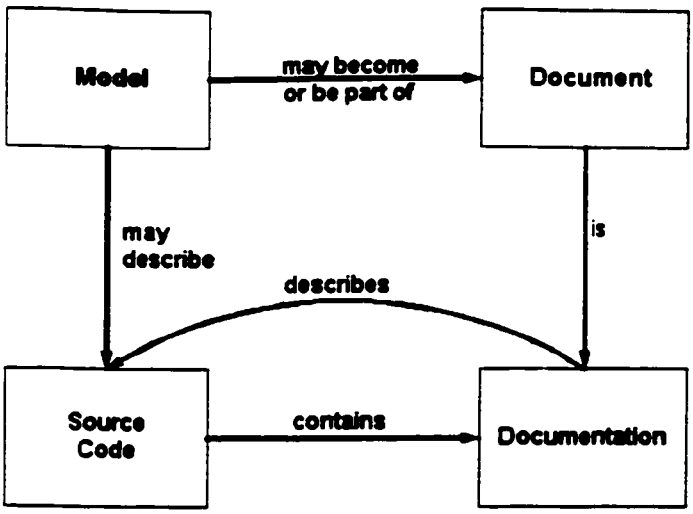

Figure 2-1: The relationship between models, documents, source code, and documentation. [1]

Ambler describes models as temporary artefacts that may or may not be transformed into a permanent document and form part of the software documentation.

We agree that some artefacts, such as software models, may have a temporary lifespan relative to other artefacts. However, our perception is that models and documents are both merely subsets of the notion of documentation as a whole. Documents need not be permanent artefacts in a software system, nor must all models be disposable. Documentation is an abstraction of knowledge about a software system and only as long as a document or model (or any other artefact) can effectively communicate knowledge does it forms part of the project's software documentation, even if only for a short lifespan. In fact, we also concede that the source code itself is one type of artefact as it, too, conveys knowledge about the software system.

2.1.3 The Role of Documentation

Bill Thomas, Dennis Smith and Scott Tilley [51] raise several fundamental questions in their discussion about software documentation. Our work will attempt to shed light on the following questions from Thomas et al.:

- What types of documentation does a software engineer (or support staff member) need?

- Who should produce, maintain and verify documentation to assure an appropriate level of quality?
- Why should documentation be produced at all?

Benson H. Scheff and Tom Georgon [45] have one perspective on some of the questions above. They believe that software engineers "do the software engineering part" [45], while documentation 'systems', which consist of support staff and automated tools, do the rest.

According to Scheff and Georgon's perspective, the role of the software engineer with regard to documentation includes:

- providing templates
- identifying the document content and format

We believe there are two inherent problems when excluding software engineers from the documentation as described above. First, although documentation templates do provide an efficient framework for re-use, the communication (and hence information) needs differ among software projects. Therefore templates will not always provide the most efficient solution based the unique needs of the software project. Second, as most concede and Abdulaziz Jazzar and Walt Scacchi affirmed in [28], much of the knowledge about a software system is in the minds of the software engineers. Based on the complications arising from communication as described by Alistair Cockburn [15], the intermediary transfer of knowledge will most likely result in lower quality documentation.

Our opinion that software engineers must be more involved in software documentation than Scheff and Georgon recommend is described below. Our opinion is based on the idea that the translation between sources, such as from an engineer to a technical writer, introduces noise into the translation and may distort the message. In particular, Scheff and Georgon's approach would involve a translation from the primary source (the software engineers) to a secondary source (the documentation systems described by Scheff and Georgon). As well, their approach would also require a translation from the documentation system to the actual documentation. As such, it is probable that Scheff and Georgon's approach would introduce a greater amount of noise into the system than a more direct approach to creating documentation.

Bill Curtis, Herb Krasner, and Neil Iscoe [17] believe that documentation should focus on how requirements and design decisions were made, represented, communicated and changed over the lifespan of a software system. As well, documentation should describe the impact of the current system on future development processes. Their study involved interviewing personnel from seventeen large software projects. Their analysis focused on the problems of designing large software systems; but many results report directly about the use (and misuse) of documentation in a software project.

Curtis et al found that behavioural factors (such as team members or other individuals of the same organization) had a substantial impact on the success of a project and were far more important than the tools or methods used by project team members. Their work describes five levels of communications involved in developing software. For our purposes, we are concerned only with their findings relating to the individual, team, and project levels (for our research we are not concerned with the company and external levels described by Curtis et al.).

Curtis et al found that most individuals indicated being frustrated with documentation and its weakness as a communication medium. They also found little evidence to support the claim that documentation reduces the amount of communication required among project team members. At the individual communication level, Curtis et al describe two main documentation vulnerabilities.

- First, there is a trade off between using verbal communication (useful for current development) and written documents for future project members.
- Second, many good software engineering communication practices were found to be impractical in a large software project environment. Curtis et al. did not explicitly list which practices were found to be impractical, but their work alluded to the idea that as a project team grows so do its communication needs; something which current communication processes cannot properly coordinate.

At the team level, Curtis et al found documentation to be inadequate at providing information to resolve misunderstandings concerning requirements or design.

Curtis et al concluded that large projects do require extensive communication that was not reduced by documentation.

Although Curtis et al focussed primarily on the communications needs of large software projects, we believe their work with regard to the effectiveness of documentation is important to projects

of all sizes: size only magnifies the problems. Our work provides statistical data that agrees with some of the issues Curtis et al identified about software documentation. We will also offer insight that may improve the ability for documentation to communicate.

Michael Slinn [49], a technical business strategist, contacted several documentation managers (including ones at IBM, Rational Systems and Sun Microsystems) asking the cost of software documentation to an organization. One manager reported, "we just assume that documentation is important, and try to do the best we can with the resources that are made available to us. We have never considered measuring the impact of the quality of work that we do on the success of the product, or the product's profitability" [49]. As opposed to merely invoking a documentation process based on assumptions, Scott Ambler [1] suggests that all documentation fulfills a purpose. If you do not have a specific purpose for a particular document (or for its maintenance, beyond just being out-dated), then you should either reconsider the need for that artefact or determine its purpose before investing resources in its production (or maintenance) [1].

Jazzar, Abdulaziz and Walt Scacchi [28] conducted an empirical investigation concerning the requirements for information system (IS) documentation. Jazzar and Scacchi's work resulted in eight hypotheses that attempt to model the requirements for achieving effective and quality documentation products and processes, stating:

> "The problem [concerning IS documentation], therefore, has been generally formulated in terms of the difficulties in developing and maintaining documents as products with a set of desirable characteristics, but without regard to the processes and organizational settings in which they are produced and consumed."

Here we note that Jazzar and Scacchi's findings indicate that a fundamental issue regarding the problems associated with software documentation have been omitted when considering process improvements. Others view the problem of creating and maintaining documentation without properly understanding the environment in which these documents are created, used and are useful to the project team. Jazzar and Scacchi's comments are in-line with our own. We believe that greater emphasis should be placed on the environment where documents exist to successfully cope with the issues revolving their use and effectiveness. These environmental factors may include how often a document is accessed and under what circumstances, how documents relate to one-another (provided this information is not inherent to the document itself) or how often documentation is maintained. Having a better understanding of how and under what

circumstances documentation is produced and consumed will help us to better understand what to offer the audience and how to maintain their attention.

Jazzar and Scacchi address the drawbacks of several proposed solutions that promote documentation traceability, standardization, consistency, conversion into hypertext and ease of document development and online browsing. According to their findings (and despite standards in documentation) documentation continues to be unavailable, inconsistent, untraceable, difficult to update and infrequently produced in a timely or cost-efficient manner [28]. Jazzar and Scacchi's findings confirm the important existence of an implicit knowledge base apart from the documented one. This undocumented knowledge base is captured in the minds of the software professionals. Jazzar and Scacchi report that when maintainers left a project, the documentation, which was once thought to be sufficient, was no longer able to address the information needs of remaining team members [28].

One might argue that all aspects of the system should be completely documented as insurance against losing the implicit knowledge base of the development team. Unfortunately, as discussed by Bill Curtis, Herb Krasner, and Neil Iscoe [17], this approach is unfeasible because of the impracticality of knowing all of the future documentation needs of project; much like the difficulty in knowing all future requirements of a software system. Curtis et al also believe such an effort will be unable to effectively communicate these future needs to maintainers. Berglund [9] and Glass [25] raise the issue about content overload, whereby having too much information reduces the overall effective of the information due to sheer size. Johannes Sametinger [44] notes that documentation is important, especially for software reuse, but traditional documentation techniques are unable to meet the needs of software professionals. Similarly, Scott Tilley [52] comments on the deficiencies of traditional software documentation techniques. Cockburn also notes that it is extremely inefficient to break down communication into units so that anyone can understand them [15] (page 19). To summarize, Cockburn offers that,

> "The mystery [about communication] is that we can't get perfect communication. The answer to the mystery [about communication] is that we don't need perfect communication. We just need to get close enough, often enough." [15] (page 19)

We agree with Cockburn's statements that the role of documentation in a software project is to offer appropriate communication to the reader, often enough.

Jazzar and Scacchi's eight hypotheses discussed above offer the following insight. Backbone systems require a more formal documentation process than non-backbone projects (H1) and require high traceability (H2). The same is true for out-of-house development compared to in-house development (H3 and H4). In-house project documentation can accommodate low consistency and completeness, but requires accurate details (H5) and less documentation (H6). Users unfamiliar with similar systems follow a different documentation process than familiar users (H7) and the unfamiliar users require documents that are more consistent, complete and traceable (H8).

H7 and H8 are fairly intuitive and are best explained by Cockburn's description of the learning process [15]; that is people follow three distinct stages of behaviour when learning; following, detaching, and fluent. Individuals begin as followers and require explicit details and instructions regarding their work. Next, these individuals detach from the prescribed process and seek alternatives when the process breaks down. Eventually, the individual becomes fluent in the skill and it is almost irrelevant whether a prescribed technique is followed or not. The fluent individual applies what is appropriate (knowing what is appropriate is part of becoming fluent) [15].

H1 to H6 recommend general documentation guidelines for software projects in terms of the documentation size (i.e the number and amount of documentation to produce), the level of traceability of the documents as well as the level of formalness in their structure and content. Our work complements Jazzar and Scacchi's as we focused our initial work on the attributes of quality documents, whereas Jazzar and Scacchi's focus was primarily on the process of quality documentation.

2.1.4 Comparisons to Other Engineering Disciplines

Often the perception of a continuous and consistent need for complete and up-to-date documentation results from comparing software engineering practices to those of other engineering domains such as civil, mechanical or electrical engineering. The argument that software documentation should always be consistent, complete and up-to-date purely because that is the practice in other engineering domains may not always be valid or appropriate in software engineering.

As Sandro Morasca points out in [39], Software Engineering is young discipline, "so its theories, methods, models, and techniques still need to be fully developed and assessed" [39] including those regarding the notion of proper software documentation. Software Engineering is quite a human-intensive domain where "some aspects of software measurement are more similar to measurement in the social sciences than measurement in the hard sciences" [39]. As a result, the need for hard documentation may be less important than having social documentation (i.e. artefacts that can communicate at a social level as opposed to merely a technical one).

Other engineering domains produce specifications from which a product is derived and usually mass-produced (either in numbers such as computer micro-chip or in size such as a office building or bridge). These specifications need to be precise; otherwise the final product(s) will not fulfil the intended purpose. Comparing this analogy to software engineering, a system's source code and build mechanism outline the specifications of the product and the compiler helps derive the mass produced product (i.e. the byte code). Software documents from an engineering standpoint are indeed complete and up-to-date in a similar fashion compared to other engineering domains. As software becomes more complex, an additional layer of more human-digestible information is required. The purpose of such a layer, as we have described in this section, is that of communication; which we argue does not imply being completely accurate or always up-to-date.

2.1.5 Perceptions of Being Out-Of-Date

Andrew Hunt and David Thomas [26], as well as several others (for instance [25], [41], and [45]), see little benefit in old documentation. In fact, Hunt and Thomas state that untrustworthy documentation poses a greater threat than no documentation at all.

Miheko Ouchi [41] shares a similar view with Hunt and Thomas. Ouchi associates being up-to-date with being correct, and conversely not-so-up-to-date with being incorrect. Ouchi then infers that incorrect content is unreliable, and being unreliable hinders the document's credibility, which ultimately reduces its effectiveness. The belief in high correlation between accuracy of content and usefulness is also shared by Glass [25], and Scheff and Georgon [45]. As well, similar thoughts about the importance of maintenance are outlined by Bill Thomas et al [51]. The underlying assumptions from these individuals appear to be that documentation is useful only to extent to which it is a correct and accurate representation of the system. Although being correct

(and hence up-to-date) are important factors contributing to relevant documentation, they are not the only factors.

Alistair Cockburn [15], as well as Scott Ambler [1] present an alternate view concerning the role of documentation. They argue that the purpose of documentation is to convey knowledge – something that can be different from merely providing information.

Alistair argues that source code presents the facts of a system and the supporting documents facilitate higher-level interpretation of those facts. A document that instils knowledge in its audience can then be deemed effective, somewhat regardless of its age and the extent to which it is up-to-date [15].

Ambler recommends to "update [software documentation] only when it hurts" [1]. Ambler ads,

> "Many times a document can be out of date and it doesn't matter very much. For example, at the time of this writing I am working with the early release of the JDK v1.3.1 yet I regularly use reference manuals for JDK 1.2.x – it's not a perfect situation but I get by without too much grief because the manuals I'm using are still good enough." [1]

To argue against constant and continual documentation maintenance, at the expense of other tasks (such as designing, constructing, testing and delivering software), Ambler further adds,

> "It can be frustrating having models and documents that aren't completely accurate or that aren't perfectly consistent with one another. This frustration is offset by the productivity increase inherent in traveling light, in not needing to continually keep documentation and models up to date and consistent with one another." [1]

Part of our work is geared towards uncovering factors beyond completeness and accuracy that contribute to a document's relevance. Similarly, we will investigate the merits of Alistair and Ambler's assertions that documentation need not be completely up-to-date or accurate to be useful.

2.2 Software Engineering Principles and Documentation

In the following sub-sections we will collect ideas from existing knowledge about software engineering practices in general and relate that knowledge to the application of sound documentation processes.

2.2.1 Avoiding Broken Windows

Andrew Hunt and David Thomas [26] present several views that define a *pragmatic programmer as*

> *one who is agile (in the sense of being early to adopt and fast to adapt to new technologies and techniques; not necessarily agile in the extreme programming sense), inquisitive, critical thinker, realistic and well-rounded [26].*

Hunt and Thomas' insight into quality software development practices and software process heuristics provides appropriate parallels that are applicable to software documentation.

For instance, Hunt and Thomas suggest avoiding broken windows. A broken window in software documentation is

> *any artefact of a software system that elicits poor design, incorrect decisions or poor code [26].*

The definition above does not imply that software artefacts must be perfect. Rather, it is important to maintain the system in a high state of quality (both internal and external). Attempts to improve areas of poor quality in a system should be a priority. Working in an environment of high quality promotes a continued level of high quality work. To align this statement with software documentation, we do not imply that a *broken window* document refers to a document that is out-dated. Instead, we believe that documentation breaks when it no longer fulfils its purpose in an efficient manner.

2.2.2 Keeping Software DRY

The DRY principle (don't repeat yourself)

that every piece of knowledge must have a single unambiguous, authoritative representation within a system [26]

is well described in the context of software construction. Techniques such as abstraction, encapsulation and inheritance all attempt to follow and promote the DRY principle. Even many programming languages including Java and .Net promote DRY principle with the idea of write one, run everywhere. Techniques, tools and technology attempt to provide similar mechanisms to facilitate the DRY principle to software documentation. Unfortunately, we believe that duplication will continue to exist in software documentation for several reasons. First, our inability to seamlessly relate knowledge among various level of information abstraction hinders our ability to effectively cross reference knowledge. For example, the translation of requirements to specifications and design, at present, will result in an overlap of information and as one changes, it is likely the other related documents will require changes as well. Second, the current impracticality of automating all processes that require duplicate knowledge suggest that duplication will be tolerated under circumstances where the effort to eliminate duplication is far greater than the effort to manage that duplication. For example, summarizing information from various sources for a presentation by cut and paste is more appropriate than parsing the information to automatically generate the report based on the existing collection of documents; especially if the presentation will only be used on one occasion. Third and related to the second issue, current tools and technologies do not always allow individuals to effectively follow the DRY principles. For example, sharing information between file formats (such as .html, .doc and .txt) is usually best achieved by duplication.

Although striving to achieve the DRY principle is important, it is equally important to accept that duplication will occur. Rather than trying to avoid duplication altogether, a more realistic goal should be to manage and minimize your project's duplication. To help follow the DRY principle, Hunt and Thomas suggest that all secondary sources of knowledge result from a derivation of the primary source [26]. Although not practical for all situations, they suggest creating small tools and parsers to manage these secondary sources. Some practical examples available to help reduce duplication include tools such as Ant to distribute your software, JavaDoc to publish the software's API and JUnitReport to publish the software's testing results. JUnitDoc is another interesting tool that builds JavaDoc API documentation, but it also integrates associated test cases based on JUnit into the API documentation.

Learning to identify and manage sources of duplication will help us to remove them, or at least to minimize their adverse effects on the software project as changes occur.

2.2.3 Good enough means more than just enough

Andrew Hunt and David Thomas [26] describe the concept of good-enough software. We have adapted their definition to align with the principles of *good enough* in regard to software documentation. Good enough documentation does not imply low or poor quality. Rather, it is an engineered solution that attempts to find a balance between forces that are in-favour and opposed to documentation. Such forces include stakeholders, future maintainers, technical limitations, budget and timing constraints, managerial and corporate guidelines as well as personal state of mind.

The notion of good-enough; or perhaps more suitably, appropriate, should be applied to the context of software documentation. For instance, and as cited in Section 2.1.4, many individuals believe that documentation is practically useless unless it is a well maintained and provides a consistent and accurate view of the system it documents. Context free, it seems obvious that up-to-date documents are better and more useful. However, when forces such as budget and timing (including the time to actually implement the software) are incorporated into the decision making process of a software project, several trade-offs are necessary. One side effect may be the extent of documentation and the extent to which it is maintained. To re-iterate, using an engineered approach to software projects, the most appropriate documentation may be documentation of perceived lower quality. Scott Ambler also concedes that documentation need only be good enough to be effective [1].

A similar point by Hunt and Thomas is to understand when to stop; to understand when a system, or in our situation documentation, is good-enough.

Another relevant observation to consider is that as a software project evolves, so too do the needs of documentation. Scott Ambler describes the issues concerning the changing needs of documentation. In particular, Ambler says that

> "during development you're exploring both the problem and solution spaces, trying to understand what you need to build and how things work together. Post-

development you want to understand what was built, why it was built that way, and how to operate it" [1].

Therefore maintaining development documentation may be counterproductive to the needs of the software team once the software reaches post-production.

Our work hopes to substantiate Ambler's claim that documentation needs change throughout a project's lifecycle. We hope to use this information to further support our claim that quality software documentation need not always be as consistent and accurate to be effective.

Cockburn warns that due to the limited resources of a software project, additional efforts to improve intermediate work products, such as documentation, beyond its purpose are wasteful. Knowing the point of diminishing returns when creating these intermediate artefacts allows individuals to better reach the point of sufficient-to-purpose. [15]

The evolution of the needs of software team members further supports the multi-dimensional considerations required when considering documentation maintenance. Alone, updated documentation is far more useful than outdated documentation; but when considering the environment in which the documentation must exist, the importance of being up-to-date is not as well understood, and perhaps is not as important a factor in software quality, or even documentation quality, as one may believe.

2.2.4 Applying Reverse Engineering Concepts to Documentation

Tools that can extract information directly from source code could be a very useful artefact for a software system.

Scheff and Georgon [45] describe the requirements for effective documentation automation. In particular, automation demands:

- Completely defined inputs
- Responses to predetermined items
- Unambiguous text
- Standard use of language and format

- Mandated output for contractual requirements

Unfortunately, their description is quite vague, and somewhat ambiguous. It is difficult to truly understand their intentions with regard to the requirements for effective software documentation.

Bill Thomas, Dennis Smith and Scott Tilley [51] provide a clearer picture of the objectives of automation in software documentation. They describe reverse engineering in the context of documentation as one way to produce, and more importantly maintain, accurate documentation.

Thomas et al [51] describe the role and objectives of reverse engineering tools as follows:

> "These [reverse engineering] tools create ancillary documentation, provide graphical views of the software system, and generally attempt to augment the knowledge hidden in the source code with secondary structures."[51]

However, Thomas et al believe such automated tools are underused. These tools could be helpful, but unfortunately little is known about what types of documentation are useful and to whom and under what circumstances. "This [observation outlined above] highlights the underlying problem with current documentation creation processes; if no one understands what is needed, it should come as no surprise that tools that produce this type of documentation are rarely used by real-world software engineers"[51].

Janice Signer et al.'s study of software professionals ([47] and [48]) provides some insight into the professional needs of software engineers. Unfortunately, Singer et al.'s work reported mixed results regarding the use of documentation. The results of their questionnaire revealed that the most cited activity of the participants was reading documentation. Yet, their follow-up studies observing software professionals as they work, Singer et al. found documentation was consulted on a much less frequent basis. "Clearly, the act of looking at the documentation is more salient in the SEs' minds (as evidenced by the questionnaire data) than its actual occurrence would warrant." [47].

Our work is focussed at better understanding the role documentation in a software project and how it is created and manipulated in a software environment. From this improved understanding, we hope to improve processes of documentation maintenance and inspire new and / or improved technologies to better automate documentation.

2.3 Software Documentation Metrics

Having the ability to rate the effectiveness of a software document using formulated techniques having quantifiable metrics is a useful tool for knowledge management because it allows us to rank documentation based on measurable attributes. Although measuring relevance is difficult due to the subjectivity of the definition. We hope that our efforts will improve our ability to predict the relevance of a document based on heuristics about relevance, both within the context of documentation as well as taken from similar domains in computer science.

2.3.1 Limitations of Readability Formulas

To date, one of the most publicized means of rating the content of documentation is the application of readability functions. These functions consider metrics such as sentence structure and length, as well as grammatical writing style, to place a grade level on a piece of writing.

In a technical environment many individuals disagree about the usefulness of such measures at predicting the relevance of document. In particular, George Klare revisited his 1963 his book "The Measurement of Readability" in Readable Computer Documentation [32]. Klare states that he is sceptical about applying readability formulas to computer documentation. Klare believes that readability formulas do not perform well for lists, tables, structured text and therefore are unsuitable to most writing found in software documentation. Klare also notes the relationship between readability and comprehension; believing that improving readability increases reading speed and acceptability but comprehension of the document is not guaranteed. Klare notes that a major problem is that there are few usability studies on the quality of documentation [32].

Janice Redish [43] expanded on Klare revised work. Redish emphasizes that readability formulas have even more limitations than Klare suggested. Redish points out that readability formulas were developed for children's school books, not technical documentation. She also notes that most readability formulas ignore the effects of content, layout, and retrieval aids (indexing for example) on text usefulness.

Redish advocates usability testing over readability formulas, even if only a few individuals are involved in the testing. Redish states that readability formulas predict only the level of reading ability required to understand a document and provide no insight in the causes or potential

solutions to the problems found in a document. On the other hand, Redish says that usability studies not only demonstrate the extent to which actual people are able to understand a document, but also are able to identify where certain flaws in the document exist.

Although readability studies may offer some insight in the usefulness of documentation, we concur with Redish's observations and beyond our acknowledgement that readabilities have been used to gauge the quality of a document: we will not incorporate these formulas in our work.

2.3.2 Applying Buffer Management Techniques to Documentation Relevance

This section highlights past work on *buffer replacement* and parallels their findings to several documentation problems.

We start by introducing the work of Björn Jónsson, Michael J. Franklin and Divesh Srivastava [29]. These individuals introduced buffer replacement techniques to improve information retrieval in 1998. They focussed on exploiting the well established buffer replacement techniques from the database community in the realm of information retrieval. Jónsson et al's work involved information retrieval at a query level and sought to efficiency avoid excessive processing on multiply-refined searches based on access patterns to similar queries. Our work is seeking to apply these buffer replacement algorithms at an even more abstract level; that is, ranking the relevance of a document against the environment in which it exists based on activity within that environment.

To select a buffer replacement technique suitable to documentation, we look to Lee et al's [34] work on techniques that combine the recency and frequency of access patterns of a particular item in a set. They show that combining the elements of recency and frequency of use greatly improves buffer management. To demonstrate the parallel of cache block replacement policies to software documentation relevance, we have extended Lee et al definition of block replacement policies [34] to apply, more generally, to knowledge management as:

> *the study of the characteristics or behaviour of workloads to a system. In particular, it is the study of access patterns to knowledge within a knowledge base and is based on recognition of access patterns through acquisition and analysis of past behaviour and history.*

Lee et al [33] introduced the Least Recently / Frequently Used (LRFU).

a buffer management policy that subsumes both the Least Recently Used (LRU) and Least Frequently Used (LFU) policies.

approach to buffer management. We will incorporate this metric to assist in our calculation of documentation relevance. We have chosen LRFU due to its ability to combine the benefits of both popular techniques, Least Recently Used (LRU) and Least Frequently Used (LFU).

The LRU approach

is a buffer management policy that ranks blocks by the recency with which they have been accessed.

LRU bases its decisions on how recently documents have been accessed. Although highly adaptable, it is criticized for being short-sighted; that is, it only considered recent history when gauging the relevance of a block. We also considered, but later rejected, an improved LRU algorithm (known as LRU-K described by O'Neil in [40]) because of the somewhat arbitrary nature of the k parameter – which is used to improve LRU's tendency to be short-sighted.

The LFU approach

is a buffer management policy that ranks blocks by the frequency with which they are accessed.

LFU is based on the frequency with which documents have been accessed. Although it is often used, LFU results can be deceiving due to the issues of locality where a document is accessed frequently in a small time frame, but overall is rarely referenced [34].

The metric used to rank documents in LRFU is called the Combined Recency and Frequency (CRF) value. A document with a high CRF values is more likely to be referenced again in the future based on the LRFU policy. To incorporate the recency of a document into the CRF value, a weighting function $F(\Delta t)$ is used, where $\Delta t = t_{now} - t_{d(k)}$ is the time span between now, t_{now}, and the time, $t_{d(k)}$, of the k^{th} access to a document, d. The frequency of a document's use is incorporated into the CRF value by adding the sequence of recency weighting function. The CRF equation is described below in Equation: 2-1.

$$CRF_{t_{now}}(d) = \sum_{i=1}^{k} F(t_{now} - t_{d(k)})$$

Equation: 2-1
CRF Value

Based on the above definition, Lee et al state that

> "In general, computing the CRF value of a block requires that the reference times of all the past references to that block be maintained. This obviously requires unbounded memory and thus, makes the policy unimplementable." [34]

As such, Lee et al transform Equation: 2-1 into a recursive equation based on the k^{th} reference to a document. This equation is shown below in Equation: 2-2.

For Equation: 2-2 to hold, Lee et al [34] state that the weighting factor, $F(\Delta t)$, must maintain the following property: $F(a+b) = F(a)F(b)$ or $F(a+b) = F(a)+F(b)$. Please refer to Lee et al's work in [34] for more information regarding the derivation from Equation: 2-1 to Equation: 2-2. Our work will use the weighting factor described in Equation: 2-3 and used by Lee et al [34] that satisfy the first property above ($F(a+b) = F(a)F(b)$).

$$F(x) = (\frac{1}{p})^{\lambda x}$$

Equation: 2-3
LRFU Weighting Function

The function above is used to weight the importance of recency and frequency with regards to predicting future access to an artefact. Varying p ($p \geq 2$) and λ ($\lambda \geq 0 \wedge \lambda \leq 1$) will allow the weighting function to exercise the full range of possible combinations of importance of recency and frequency. "An intuitive meaning of λ in this function is that a block's CRF value is reduced to $\frac{1}{p}$ of the original value after every $\frac{1}{\lambda}$ time steps." [34].

Potential future work in our field of document relevance could investigate the relationship between recency / frequency of a document's access to the likelihood of being accessed again in the near future (based on a ranking scale). This work could then provide potentially more suitable parameter values for Equation: 2-3. Because our work is only beginning to explore the potential application of buffer replacement strategies to software documentation, we will use the

standard values of $p = 2$ and $\lambda = \frac{1}{2}$ to give equal preference to the frequency and recency of a document's access.

Finally, to calculate the CRF value a particular moment in time (since not all accesses are made at the time you wish to consider), you apply Equation: 2-4. This equation is used to relate the CRF value of an item's last access to its present value.

$$CRF_{t_{now}}(d) = F(t_{now} - t_k)CRF_{t_k}(d) \qquad \text{Equation: 2-4}$$

Updated CRF Value

For example, Figure 2-2 shows the CRF value of a document, d, at time 6 that was accessed at time 1, 3, and 4. To avoid complicating the process of calculating CRF, we used $p = 2$ and $\lambda = 1$ for the example in Figure 2-2.

$$CRF_{t1}(d) = F(0) = 1$$
$$CRF_{t3}(d) = F(0) + [F(3-1)] \times CRF_{t1}(d) = 1 + F(2) \times 1 = 1 + \frac{1}{4} = \frac{5}{4}$$
$$CRF_{t4}(d) = F(0) + [F(4-3)] \times CRF_{t2}(d) = 1 + F(1) \times \frac{5}{4} = 1 + \frac{1}{2} \times \frac{5}{4} = \frac{13}{8}$$
$$CRF_{t6}(d) = [F(6-4)] \times CRF_{t4}(d) = F(2) \times \frac{13}{8} = \frac{1}{4} \times \frac{13}{8} = \frac{13}{32}$$

Figure 2-2: An example LRFU calculation

Lee et al [34] have demonstrated that LRFU successfully avoids the limitations of LRU and LFU while exploiting both of their benefits. LRFU is able to reduce the effects of locality without limiting the referenced history. Finally, LRFU algorithm can be adjusted based on the relative importance of recency versus frequency [34]. For the many reasons cited above, our work with software documentation relevance will incorporate the LRFU algorithm.

The use of buffer replacement strategies will help identify the usefulness of a document beyond the content contained within it and will take into account how and when the document is used. This environmental information should improve our ability to locate appropriate information and better prioritize maintenance; much as the application of buffer replacement strategies have accomplished in other similar domains such as information retrieval and database management. Please refer to Chapter 6 for more details about using buffer replacement techniques in predicting documentation relevance.

2.3.3 Authoritative Techniques for Rating Documentation Relevance

Much of the current work regarding ranking and prioritizing information is in regard to exploring the World Wide Web (www). Reflecting on the progress made in this area, and the justifications for certain approaches to predicting the relevance of a document (or page as it is referred to in a www context) should help improve our ability to rank and retrieve information on a smaller scale, such as that present in a software project [36].

Prior to the expansion of the World Wide Web, the work of Botafogo et al. [10] regarding page relevance was focused on stand-alone hypertext environments. Botafogo's work defined the terms index and reference nodes. An index node is a node whose out degree is significantly higher than the average. A reference node is one whose in-degree is much larger than the average. Botafogo et al considered measures of centrality and authoring pages based on distances between nodes in the hypertext web. Much of their work has been incorporated into improved techniques for using the in and out-degree of a node to predict relevance.

Sergey Brin and Lawrence Page [12] implemented Google, the first search engine that successfully based search results on the link structure of the Web. Brin and Page determined the priority of a page by its page rank,

> *"an objective measure of its citation importance that corresponds well with people's subjective idea of importance." [12]*

As opposed to averaging merely the in and out-degree of the link structure, as done by Botafogo et al. [10], the Google project also weighted each reference; therefore being associated with relevant pages (for instance pointing to it, or being pointed by it) increases the weight of that reference and hence increases the rank of that page.

Brin and Page also found that prioritizing search results of a text search based solely on the titles of the pages combined with Page Rank approach to prioritize the results performed well.

To define the page rank $PR(A)$ of a page A, we consider all pages, $T_1,...,T_n$, that reference A where the total number of references to A is $C(A) = n$. The page rank calculation is shown below in Equation: 2-5.

$$PR(A) = (1-d) + d[PR(T_1)/C(T_1) + ... + PR(T_n)/C(T_n)]$$ Equation: 2-5

Page Rank

To maintain the page rank as a probability distribution, a damping factor, $d \in \{0 \le d \le 1\}$, usually set to 0.85, is used [12]. The results can be calculated using an iterative algorithm where the solution corresponds to the principle eigenvalues of the normalized link matrix. Brin and Page describe page rank as the probability that a random user will access a particular page, and the damping factor is the probability that a user will eventually leave that page.

Jon Kleinberg [33] approaches page relevance from a similar approach to Brin and Page [12], but instead of considering page rank, he considers the authority of a page. Kleinberg investigates the relationship between index and reference nodes (Kleinberg refers to these items respectively as hubs and authorities), building on Botafogo et al's [10] work to cope with a larger document space.

Jon Kleinberg, like Botafogo et al. [10] (as well as others including [12] and [2]), show that the nature of a hyperlinked environment can be an effective means at discovering authoritative sources. Kleinberg's study included searching for, and finding relevant results for terms such as 'Harvard', 'censorship', 'Java', 'Search Engines', and 'Gates'. The focus of their work was to improve the process of discovering authoritative pages based on a subset of the available pages (such as the subset returned by text based search).

Kleinberg's approach complements, as opposed to discounts, text based searching techniques showing that using the results of a text based search (to narrow the result space) as well as the information about the interconnections between documents, almost exclusively, one can produce relevant discoveries of authoritative sources.

In a similar fashion, the page rank algorithm studies little more than the page titles and the anchor titles (the text that describes a particular link). Kleinberg justifies using the in-degree and out-degree of a hyperlinked environment for several reasons. The most prominent justification for using the link structure to determine authority is that using links avoids the problem whereby a prominent page may not be sufficiently self-descriptive. Without considering link structure, these unselfish pages may not rank as highly compared to less authoritative sources (which are more self pronounced). Another justification for using link structures is the fact that text based searches produces a high number of irrelevant results [33].

Kleinberg's approach requires an initial root set of results that contain some relevant results. This root set was obtained by taking that top 200 results from a popular search engine. The set is then expanded to a base by adding documents that are either referenced by a root set document, or a document that references a document in the root set. Figure 2-3 illustrates the expansion of a root to a base set.

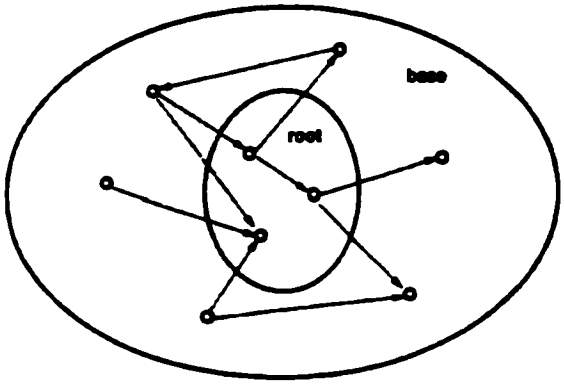

Figure 2-3: Expanding the root set into a base set [33]

For practical reasons, Kleinberg allowed each root to contribute at most 50 new documents to the root set.

Kleinberg's approach is unique in that it attempts to balance relevant sources and popular ones. Kleinberg uses the notion of authority and hub pages. A good authoritative page is referenced by good hub pages; whereas good hubs link to good authoritative pages. Understanding and tracking the quality of hub pages helps to better reflect to the quality of authoritative ones.

This mutually reinforcing relationship between hubs and authorities is represented in Equation: 2-6 and Equation: 2-7.

$$authority = \mathrm{I}^{(p)}(hubs)$$

Equation: 2-6
Authority Scores

$$hubs = \mathrm{O}^{(p)}(authority)$$

Equation: 2-7
Hub Scores

Where the vector / matrix $authority = [x_1, \ldots, x_n]^T$ is the authority score for pages 1 to n and $hubs = [y_1, \ldots, y_n]^T$ is the hub score, where $\mathrm{I}^{(p)} = L^T$ and $\mathrm{O}^{(p)} = L$. Finally, L is the adjacency

matrix of the graph outlining the references among pages 1 to n. Much like the Page Rank algorithm, an iterative algorithm will result in the principle eigenvalues of the normalized authority and hub matrix. Figure 2-4 illustrates the process of calculating the authority and hub values a page.

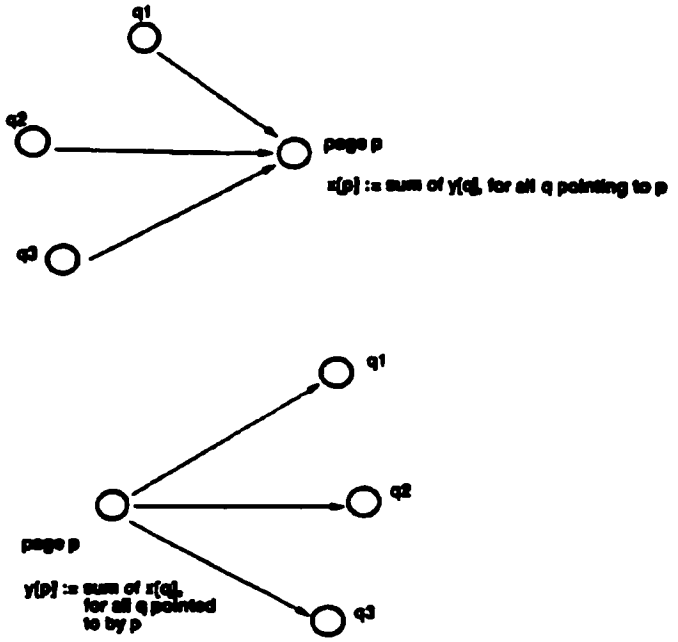

Figure 2-4: The basic operation of authority and hub value [33]

Kleinberg's study found that the authority and hub values converge relative quickly after about 20 iterations.

Signer et al., in their study of software professionals [47] and [48], found that searching was the activity most often performed. As observed by Brin and Page [12], a simple text-based search (involving only page titles and link anchors) combined with a priority algorithm based on the link structure can result in relevant results. As such, we deem Kleinberg's algorithms most appropriate in the context of software documentation. Combining a simple text based search to uncover the root set, Kleinberg's equations can then expand the possible results to a base set. Using his authoritative technique to rank documentation, the results should be relevant based on the users needs. Kleinberg's approach is also able to rank documents based on quality of the references they provide (gauged by having a high hub values). Not only can we use Kleinberg's approach to predict highly relevant pages; we can also use it to predict highly indexed pages (that is, pages that index several relevant pages).

CHAPTER 3: SURVEY ON DOCUMENTATION

A survey of software professionals was conducted as part of our research and is referenced throughout this thesis. This chapter highlights our motivation, research methods, participants demographics as well as an analysis of the questions and preliminary results of the survey. Refer to Appendix A to Appendix E for a listing of the survey questions and resulting data.

3.1 Survey Motivation and Objectives

Our motivation to conduct a survey of software professionals was to shed practical insight on the objectives outlined in Chapter 1, Section 1.2.

Tim Lethbridge, Susan Sim, and Janice Singer [37] describe software engineering as

> "a labor-intensive activity. Thus to improve software engineering practice, a considerable portion of our research resources ought logically to be dedicated to the study of *humans* as they create and maintain software." [37]

In search of answers, we performed a systematic survey to question the thoughts of software practitioners and managers. Our approach is to build theories based on empirical data; possibly uncovering evidence that questions our intuition and common sense about documentation and its role in software engineering.

The research objectives of the survey are to determine what makes software documents useful. As well, we will investigate how a document's relevance can be measured using not only its own attributes, but also compared against other documents of the software system.

Our survey investigated several factors that may affect the use and usefulness of software documentation including:

- The technology used to create / maintain / verify software documentation.
- The software domain and size.

- The experience of the development team.
- The role of the management and development team with regard to documentation.
- The styles, standards and practices involved in the documentation process.
- The turnover time between software project changes to updates in related documents (i.e. the effectiveness of documentation maintenance).
- The technologies used to automate the documentation process.

By exploring how these elements contribute to a document's relevance (by aiding or potentially hindering it), we hope to discover results that will help improve the entire documentation process.

3.2 Pilot Study

Before conducting the main survey described in this paper, we conducted a pilot study to help develop and refine the questions.

The pilot-study participants were sampled from a fourth year software engineering course offered at the University of Ottawa in the winter 2002. Most participants had some experience in the software industry.

The official survey, conducted in April 2002, featured fewer and more concise questions with an improved sampling approach that is described in the following subsection. All participants had at least one year of experience in the software industry; several had over ten years experience.

Individual responses and identifying information have been withheld to protect confidentiality. The University of Ottawa's Human Subjects Research Ethics Committee approved the conducting of the survey.

3.3 Research Methods

The survey was available online at [21] and is outlined in greater detail in appendices Appendix A to Appendix D. Bill Kalsbeek's work in [31] provided insight in our survey research methods described below. The survey introduction outlined our research goals and objectives.

Participants could then log into the system creating a user id and optionally providing a password, or continue anonymously. In creating a user id, individuals could save their responses and return at a later date to complete the survey. Participants were then asked to read and agree to our Informed Consent statement if they wished to continue. Once the consent was complete, the participants were able to answer the survey questions.

The data were stored in a database and exported to a spreadsheet application for subsequent analysis.

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

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

Most participants completed the survey using the Internet [21]. A few replied directly via email. There were a total of 48 participants who provided responses that were complete and contained valid data.

3.4 Participants

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

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

- Agile. Twenty-five individuals that somewhat (4) to strongly (5) agree that they practice (or are trying to practice) agile software development techniques, according to Question 29 of the survey.

- Conventional. Seventeen individuals that somewhat (2) to strongly (1) disagree that they practice agile techniques, or indicated that they did not know about the techniques by marking 'n/a' for not applicable.

Our rationale for the above division is that the proponents of agile techniques promote somewhat different documentation practices from those recommended in conventional software engineering methodologies. The dominant characteristics of an agile participant include test-first development strategies, pair programming, little up-front design and strong communication bonds with other developers, managers and customers. There is somewhat of an unfounded stigma that agility assumes documentation is not useful and should not be a component of a software project. Some have suggested that the participants identifying themselves as agile are merely those individuals that avoid documenting their work. The suggestion that the agile participants are merely lazy in their work-habits towards documentation is unlikely. The primary message of agility is to do what is most suitable assuring that an appropriate safety-net (i.e. test-first strategies) is available to support change. The decision to allocate fewer resources and less effort to documentation is not one of lack of effort, but rather one of cost-benefit between developing software and describing it [1].

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

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

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

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

Figure 3-1 outlines the number of participants in each category outlined above. The number of participants not included in a particular category is also highlighted. For instance, project leaders, software architects and technical writers comprise the 19 individuals not considered in the Manager / Developer categorization.

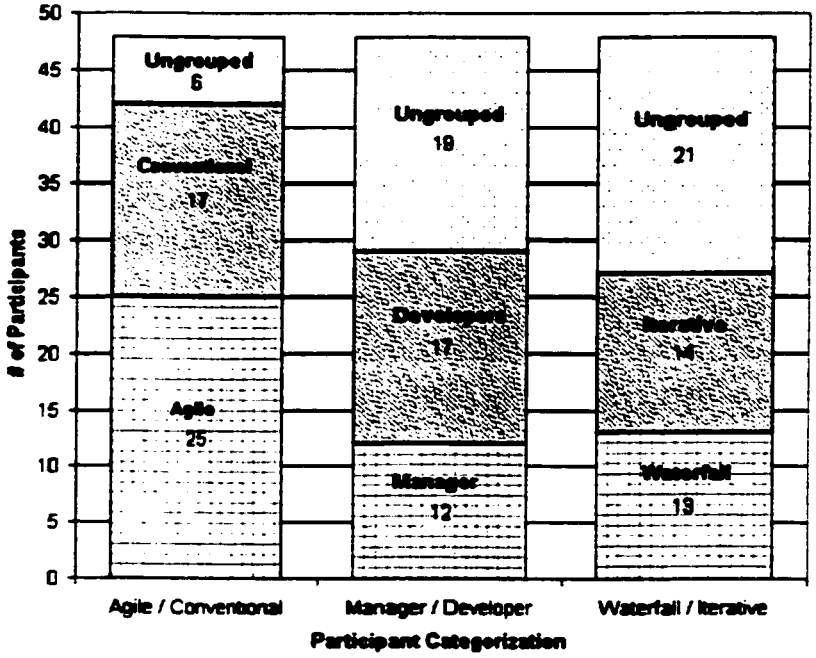

Figure 3-1: Survey participants' arranged by category

The following observations were made with respect to the category numbers shown above.

- There were slightly more agile participants than conventional participants. The individuals identified as agile agreed that they practice or they are trying to practice agile techniques. These individuals were not necessarily involved in strictly agile projects, but rather applying agile techniques in their personal software habits. Please refer to Figure 3-5 for more details about the software processes of the participants.

- Nearly half of the participants were not considered as developers or managers. It is important to note that software architects, project leaders, quality assurance individuals and technical writers comprise the individuals not included in the Manager / Developer group. Please refer to Figure 3-4 for more details about the participant's employment duties.

- Several individuals are practicing software processes other than the conventional waterfall and incremental / iterative approach. Please refer to Figure 3-5 for more details about the other software processes of the participants.

3.5 Demographics

This section will describe the participants' demographics. The divisions separate individuals based on software experience, current project size and software duties. The purpose of this section is to show that the survey was broadly based, and therefore more likely to be valid in a wide variety of contexts.

Figure 3-2 illustrates the participant's experience in the software field (based on number of years in the industry). Please note that one participant did not answer this question and is not considered in the percentage calculations.

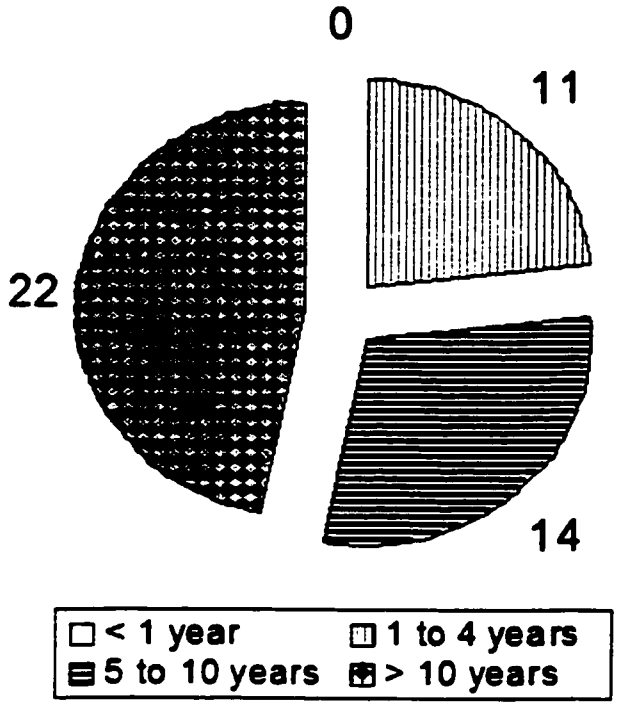

Figure 3-2: Participants' software experience in years

Table 3-1 highlights the participants' current project size. The size is estimated in thousands of lines of source code (KLOCs). Please note that two participants did not answer this question and are not considered in the percentage calculations.

Table 3-1: Project sizes in thousands of lines of code (KLOC)

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

Question 43 asked the participants what type of software products their company developed. The participants' answers varied from web application, credit card services, governmental services, medical software, to real-time embedded systems.

Figure 3-3 shows the experience the participants had with the documents listed throughout the survey based on Question 1. At least 50% of all participants had some experience with all types of documents.

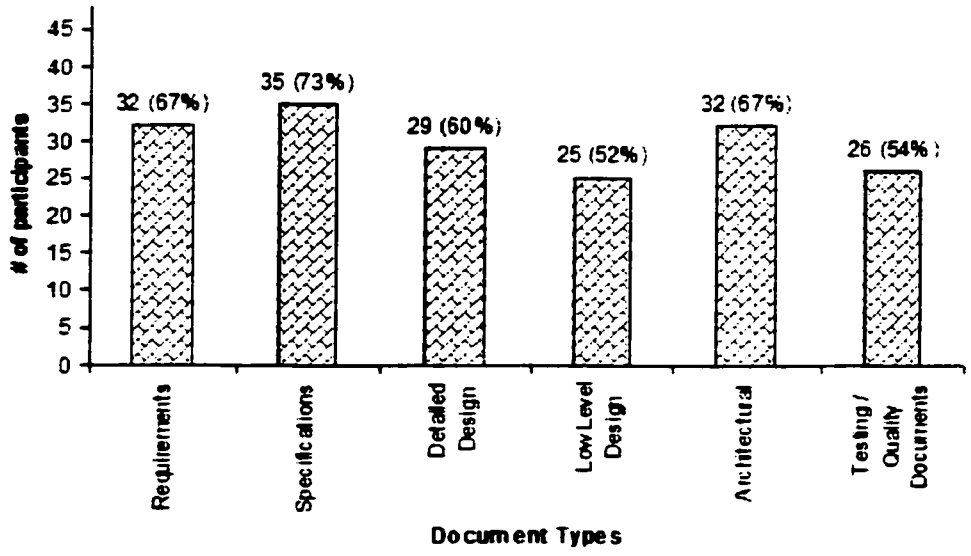

Figure 3-3: Participants' experience writing / editing / verifying software documents

Table 3-2 shows the current job functions (at the time the survey was conducted) held by the participants. Please note that many individual held several job functions.

Table 3-2: Participants' employment in the software field*

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

* Note that many participants performed one or more function.

Figure 3-4 provides a more detailed account of participant's duties by highlights the multiple roles of a participant. The overlapping employment duties are shown as subsections of a particular duty. For example, two individuals identified their roles as both Manager and Architect. This duplication is reflected in both the 'Manager' and 'Architect' columns.

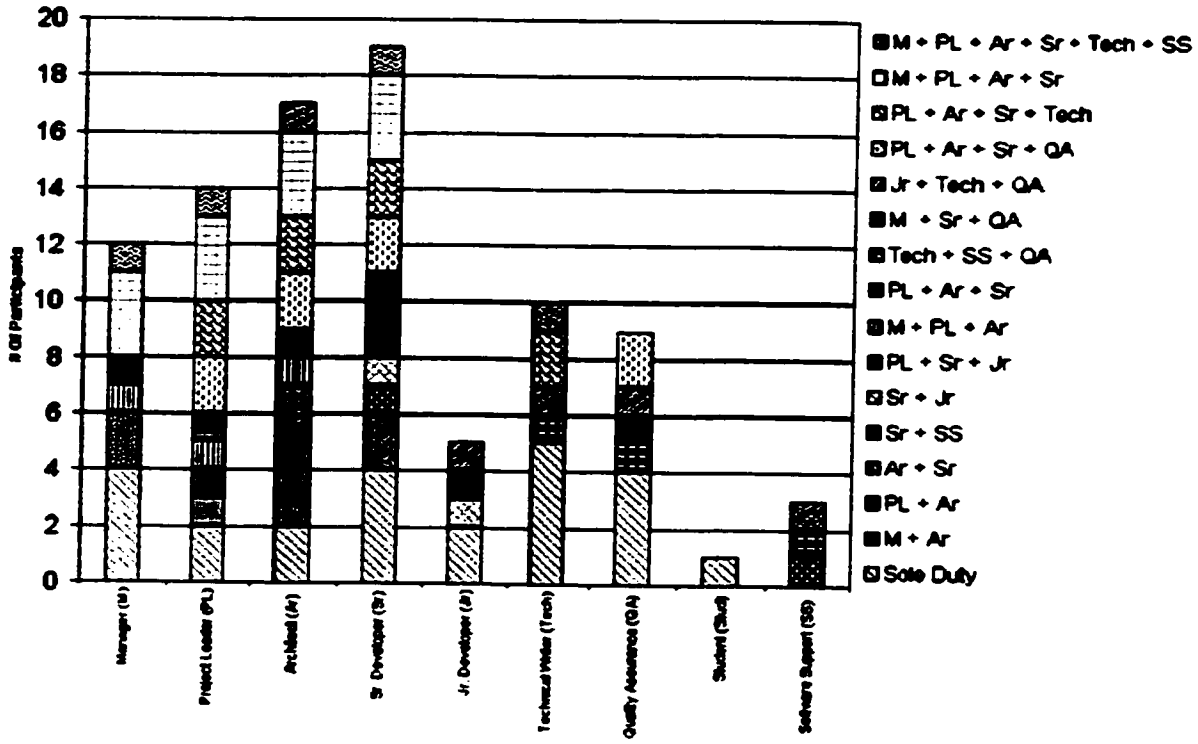

Figure 3-4: Participants' employment duties

It appears from the above data that most employment areas in the software field have been well represented. The two somewhat under-represented categories are junior developers and software support. Junior developers and software support may use documentation quite heavily but typically have less experience with software documentation their under-representation should have little effect on the interpretation of the survey results.

This survey was not directed at students, hence there were few student participants, since students are more likely to lack the software and documentation experience to provide useful results.

Below in Figure 3-5, we see the distribution of recommended software processes by a participant's management (or the by the company itself). Please note that several processes might be recommended to a particular participant.

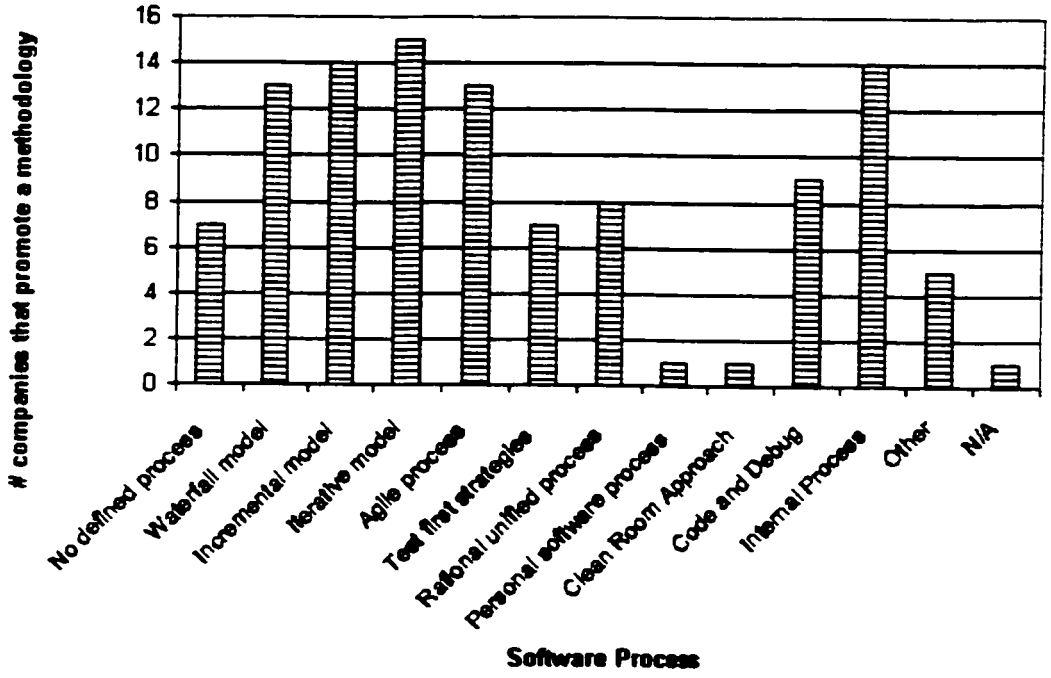

Figure 3-5: Recommended software processes by participants' management or company

There exists an even distribution between all well-known processes including the Waterfall, Incremental, Iterative, as well as Agile models. Other somewhat popular processes include Test-First and Rational Unified Process (RUP).

It is interesting to note that several individuals indicated that they follow internal processes. These processes are most likely combinations of the well-known models listed above. Another surprising element in the figure above is the fact that several participants either use no defined process or simply develop software using the style of code-then-debug. This fact is important to assure that the broad spectrum of projects was considered in this survey.

3.6 Project Size Independence

This section provides evidence that the conclusions drawn throughout the following sections (and subsequent chapters) appear to be independent from the project size (based in thousands of lines of code, KLOCs). That is to say, we believe that the findings and recommendations hold true regardless of the project size being considered.

Independence will be demonstrated by showing that participant of all categories were represented. As well, we will show low correlation between the participants' personal style of software development (between conventional and agile) and their role (managerial to developer) as demonstrated by the correlation coefficient around zero.

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

Table 3-3 illustrates the project size distribution for all categories outlined in Section 3.4.

Table 3-3: Participants' project size in thousands of lines of code (KLOCs)

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

We would like to point out that a larger than expected portion of agile participants are working on large projects (agile development is typically associated with small projects). Our phrasing of practicing agile techniques helps explain this high percentage. In the context of our research, individuals were asked if they practice agile techniques. Agreement with this statement does not necessarily imply that the project itself is agile. As such, it is not unfounded to have such a large portion of agile techniques applied to large projects.

Using Spearman's Rank Correlation [30], the correlation between project size and the individual's software techniques (ranging from highly conventional to highly agile) was very low (-0.09). Similarly, the correlation of project size to the individual's role (ranging from highly managerial to highly developmental) was quite low (0.19).

The low correlation above, and the fair representation of the software categories outlined in Section 3.4 suggest that the results should hold regardless of project size.

3.7 Question Topics

The survey consisted of 50 questions including multiple-choice, short answer, ratings, and free-form questions.

The question topics included:

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

3.8 Question Formats

This section outlines the types of questions that were asked in the survey and how they were presented to the participants.

Likert Scale Questions: The participants were asked to rate to what extent they agreed or disagreed with various statements. Ratings were scored as follows: strongly disagree (1), somewhat disagree (2), indifferent (3), somewhat agree (4), and strongly agree (5). The Likert Scale questions were 13 – 35.

Rating Questions: The participants were asked to rate several items based on a 5-point scale. The scale values varied for each question but the ascending nature ranging from least / worst (1) to most / best (5) was consistent throughout all questions. Participants could also abstain from the

question by leaving it blank or selecting N/A for not applicable. The Rating questions were 5 – 9, 12, 38 – 40 and 48.

Multiple Choice Questions: The participants were asked to choose an appropriate answer based on the available answers provided. The survey included single-answer (question numbers 41 and 42) as well as multi-answer (question numbers 1-4, 44, 45, and 46) multiple choice questions.

Open Ended Questions: The participants were able to answer these questions in a free-form nature. The open ended questions were 10, 11, 36, 37, 43, 47, 49, 50.

3.9 Questionnaire Analysis

The following sub-sections will group and analyse the survey questions. As well, these sub-sections will provide a preliminary glimpse at the survey results. In subsequent chapters, the data will be analyzed and inferences drawn based on the survey findings.

3.9.1 Current State of Software Documentation

This section outlines the questions to and results of several questions concerning the participants' own experience with software documentation.

Question 14 asked if the participants felt that documentation is important, but not that useful in their current organization. It is important to differentiate between individuals that have a strong aversion to documentation and those that believe under present situations the documentation is not that useful (but still important). Table 3-4 shows that many individuals strongly disagreed with Question 14, implying that most participants do believe that the documentation in their current organization is useful.

Questions 15 and 16 asked if the software documentation available to participants is easy to understand, easy to cross-reference, brief and to the point. From Table 3-4 below, we see that most documentation is both easy to understand and navigate as well as brief and to the point.

Question 17 asked the participant if the appropriate documentation was easy to locate when required. Conversely, Question 34 asked if the appropriate documentation was difficult to find and navigate due to the large number of documents available. Improved quality and usefulness can be achieved not only by the content of the information, but also the ability to locate the appropriate information [3], and [4]. Table 3-4 below shows a strong split between documentation being easy to locate or not, but a high portion of individuals believe that having too much documentations hinders their ability to efficiently locate the appropriate documentation. The implications of these results are discussed in detail in Chapter 4 Section 4.5.

Table 3-4: Current ease of use of software documentation

Question	Mean	St. Dev	Mode	Strongly Disagree (1)	Somewhat Disagree (2)	Indifferent (3)	Somewhat Agree (4)	Strongly Agree (5)	n/a
14 - not useful	2.38	1.40	1	17	10	4	9	4	2
15 - easy to understand and navigate	3.38	1.29	4	4	12	2	19	9	0
16 - brief and to the point	3.30	1.04	4	2	12	5	25	2	0
17 - easy to locate	2.79	1.12	2	5	18	5	17	1	0
34 - too many documents available	3.11	1.29	4	7	9	4	21	4	1

Question 20 asked if documentation is always outdated. Question 4 asked the participants how long it takes for the support documentation to be updated as the software system undergoes changes. Question 21 asked if documentation can be useful even if it might not always be up to date. As discussed in Chapter 3, many individuals believe that being up-to-date greatly affects documentation quality. Question 21 will provide evidence to help confirm or refute the importance of being up-to-date and being useful. Table 3-6 confirms the perception that documentation is always out-dated, and the results in Table 3-5 suggest the most likely reason for being out-dated is that documentation maintenance rarely occurs. The interesting data in Table

3-6 shows that out-dated content can still be useful. A more detailed analysis of the above is available in Chapter 4, Section 4.3 and Section 4.4.

Table 3-5: How often are the following documents updated when changes occur?

Question 4	Mean	St. Dev	Mode	Never (1)	Rarely (2)	Months (3)	Weeks (4)	Days (5)	N / A
Requirements	2.78	1.22	2	2	14	2	6	3	0
Specifications	3.12	1.13	2	0	11	3	8	3	2
Detailed Design	2.96	1.27	2	2	10	4	5	4	2
Low Level Design	2.96	1.36	2	2	11	0	6	4	2
Architectural	3.04	1.15	2	1	10	5	7	3	1
Quality Documents	3.74	1.25	5	1	3	6	4	9	3

Question 22 asked if most software documents have a finite useful lifetime and should be subsequently discarded (or removed from the primary document repository). It is well known, [25], [41], [51], and well supported by the data above, that documentation is rarely maintained. It is also believed that preserving too many documents can hinder documentation quality on a whole [25], [15]. This question will help uncover the participant's comfort level regarding disposable documentation, that is, creating documentation for a particular purpose with no intent to keep it once the purpose is accomplished. Once that purpose is fulfilled (potentially transferring the knowledge from the disposable document into other more permanent areas of the system, such as the source code), the document may be safely discarded.

Question 27 asked if documentation that the participants found useful during inception / construction of a software project differs from that which they found useful during maintenance and testing. Question 27 is similar to Question 22 in that it is interested in understanding the information dynamics of a software system.

Table 3-6 shows a split of opinion as to whether documents have a finite lifetime (after which they should be discarded). But, we also see that most participants believe that their documentation needs evolve over the lifespan of project. Interesting inferences about these results are highlighted in Chapter 5, Section 5.2.

Question 35 asked if the participants' believed that only a few software documents are useful to them. From Table 3-6, we see that most people somewhat believe that only a few documents are useful to any individual.

Table 3-6: Current state of software documentation maintenance

Question	Mean	St. Dev	Mode	Strongly Disagree (1)	Somewhat Disagree (2)	Indifferent (3)	Somewhat Agree (4)	Strongly Agree (5)	n/a
20 - always outdated	3.60	1.23	4	2	11	1	19	11	1
21 - useful even if not up to date	3.96	0.93	4	0	6	3	24	13	0
22 - finite useful lifetime	2.89	1.37	2	8	15	2	15	6	0
27 - usefulness differs by SE phase	3.83	1.20	4	3	5	2	21	15	0
35 - few documents useful	3.48	1.22	4	4	7	7	18	9	1

Question 6 asked how often the participant consulted the available software documentation when working on that software system. The participants rated between (1) for never and (5) for always consulting the documentation. The results are summarized below in Table 3-7.

Table 3-7: How often is documentation consulted

Question 6	Mean	St. Dev	Mode	Never (1)	Rarely (2)	Occasionally (3)	Often (4)	Always (5)	N / A
Requirements	3.14	1.41	4	5	5	4	9	5	0
Specifications	3.85	1.29	5	2	2	5	6	11	2
Detailed Design	3.19	1.36	3	3	5	9	2	7	2
Low Level Design	2.96	1.31	3	4	5	8	4	4	2
Architectural	3.68	1.36	5	3	2	7	5	11	0
Quality Documents	3.28	1.40	3	4	3	6	6	6	3

Question 5 asked how effective certain types of software documents are in reflecting the true state of a software system. The participants gave ratings between one (1) as completely useless and five (5) as extremely effective. The results are summarized below in Table 3-8.

Table 3-8: How effectively can documents reflect the true state of a software system

Question 5	Mean	St. Dev	Mode	Completely Useless (1)	Somewhat useless (2)	Indifferent (3)	Software effective (4)	Completely effective (5)	N / A
Requirements	3.11	1.26	3	4	4	9	7	4	0
Specifications	3.50	1.03	4	0	5	8	8	5	2
Detailed Design	3.19	1.13	3	2	5	8	8	3	2
Low Level Design	3.00	1.18	2	2	7	7	5	3	3
Architectural Quality	3.54	1.04	4	1	4	6	13	4	0
Documents	3.67	1.27	5	1	5	3	7	8	3

Question 50 asked the participants to describe their company's policy for software documentation as well as to what extent and how successfully it is followed. Selected responses to Question 50 include,

- There is no policy.
- The documentation should be usefu[l] otherwise it is obsolete. This is not followed very closely since no one is responsible for removing obsolete documentation ...
- Document everything. Have it reviewed and signed off once completed.
- We are supposed to give program documentation to a change management team so they can keep it in a central place. There are no guidelines of what needs to go into the documentation. In practice we give them a subset of our documentation (usually just a jarred up version of our javadoc). Often times we will hand them the same set of documentation that we did last install and they never seem to notice.
- Documentat[ion] is part of the development processes and it is part of development cycle through out the p[ro]ject.
- My company doesn't have a fixed documentation policy.

3.9.2 Existing Documentation Tools and Technologies

This section highlights survey questions relating to the current use of documentation tools and technologies. The following section will outline the participants' view of the merits of potential technologies.

Question 18 asked if tools to view and browse software documentation were bulky and inefficient. In addition, Question 19 asked if these same tools facilities the participants work. Question 28 asked if the effort required to format documentation was greater than that of providing the document with content.

Table 3-9: Participants' view of existing and potential documentation technologies

Question	Mean	St. Dev	Mode	Strongly Disagree (1)	Somewhat Disagree (2)	Indifferent (3)	Somewhat Agree (4)	Strongly Agree (5)	n/a
18 - tools bulky and inefficient	2.71	1.24	2	7	15	10	7	5	2
19 - tools facilitate work	3.73	1.03	4	3	1	10	21	9	2
28 - formatting work required is excessive	2.66	1.48	1	14	11	4	10	7	0

Table 3-9 above suggests that most participants believe that documentation tools facilitate their work and that these tools are also neither too bulky nor inefficient. Some believe that the effort required to format documentation is excessive, but most disagreed. The implications of the results above are discussed in detail in Chapter 5, Section 5.1.

Question 49 asked if the participant uses tools to automate software documentation (and to what extent are these tools utilized). It is also important to understand the current practices of documentation automation to help predict the usefulness of potential automated tools.

Selected responses to Question 49 include,

- Rational SODA. We try to use it with mixed results.
- JavaDoc for Java code documentation
- Together ControlCenter for reengineering models
- [D]oxygen / 100% of class-documentation
- No, these tools seldom provide intelligent documentation.

Question 10 asked what types of software artefacts the participant finds most effective, whereas Question 11 asked which types are least effective for software documentation.

Selected responses to Question 10 indicating examples of the most effective documentation artefacts,

- Plain text for requirements from customers. Most of the rest is useless.
- [D]ifferent use at different lifecycle points; no short answer. Personally, I like a combination of domain object models and high-level sequence diagrams, tied to use cases.
- Case maps, sequence diagrams and finite state machines are all good as they show what to be expected next. In most cases, they can be verified quite easily.

Selected responses to Question 11 show examples of the least effective documentation artefacts,

- Use case diagrams are useless unless they can be linked to the use case itself.
- Inaccurate documentation and use cases when the use case is already implemented.
- Detailed designs. These are expensive to create and maintain because of their low level, and are not kept in synch. Source code with a powerful IDE that allows you to navigate between related objects is always better.
- Flowcharts and railroad-tracks [presumably, the writer is referring to sequence diagrams]. Reason: anything even slightly complicated becomes unreadable and unmodifiable.

- Ones that were created at the direction of management instead of from the need of the actual audience for the artefact.

Question 36 asked what tools the participant finds most helpful, whereas Question 37 asked which tools are least helpful to create / edit / browse and generate software documentation.

Selected responses to Question 36 highlighting useful tools include,

- JavaDoc, Word, Rational Rose
- JavaDoc and Toghether Control Center
- JavaDoc and any word processor. All our documents have a simple indent and number layout. For diagram I prefer Visio, because it can make a variety of diagram types.
- Nothing special, just any editor.

Selected responses to Question 37 highlighting not-so-useful tools include,

- Any sort of requirements management software.
- Microsoft Word.
- Excel and powerpoint. I've used them but it was difficult.
- Rational Rose

Chapter 5 Section 5.1 provides a more in-depth look at the results to the questions above.

3.9.3 Potential Documentation Tools and Technologies

This section highlights survey questions relating to the merits of potential documentation tools and technologies.

Question 24 asked if software documentation contained a lot of information that could be extracted directly form the source code. Question 25 asked if tools that help extract information from source code are powerful for creating documentation. It is important to understand if participants believe in documentation extraction from source code and documentation automation.

Question 30 asked if automated testing is a useful tool for software documentation. Software testing helps exhibit the true state of the system, and such an effort might be a useful means of documentation.

Question 13 asked if graded software documentation (based on factors such as length, accuracy, ease of use, etc) would increase its usefulness to the participant. Past research [32], [42] has questioned the usefulness of readability formulas with respect to computer documentation. This question is important to determine if factors beyond readability can be effective in grading the usefulness of software documentation.

Question 31 asked if tools to track changes in a software system for the purpose of updating and maintaining documentation would be useful. Several individuals believe that documentation maintenance is a key factor in a project's success. Unfortunately, it is rarely the case that documentation is well-maintained. Tools that facilitate maintenance are important in a software project. This question attempts to validate tracking of software project changes for the purpose of document maintenance. It is important to note that a change need not only be reflected by a change to the system's source code. Rather, a change could be a change to any project artefact such as a requirements or specifications document.

Table 3-10: Participants' view of potential documentation technologies

Question	Mean	St. Dev	Mode	Strongly Disagree (1)	Somewhat Disagree (2)	Indifferent (3)	Somewhat Agree (4)	Strongly Agree (5)	n/a
24 - info could be extracted from source	3.38	1.33	4	5	10	4	17	10	0
25 - source extractors are useful	3.76	1.19	5	2	6	7	14	15	2
30 - automated testing facilitates documentation	3.70	1.11	4	2	5	4	19	9	7
13 - value of grading	3.37	1.29	4	6	4	10	16	9	0
31 - tools to maintain documents useful	4.13	1.00	5	2	1	5	18	19	0

Table 3-10 suggests two main points. First, most participants believe that more information can be extracted from source code to produce documentation. Second, most participants agree that

grading documentation, as well as tracking changes to the software system would be useful. Further analysis of the data above is discussed in Chapter 5 Section 5.3 and Section 5.4.

3.9.4 Software Documentation Responsibilities

This section outlines the participant's perception of who has the responsibility for certain types of documentation tasks. The participants were asked to select which documentation role (such as a manager or developer) was most appropriate for a particular documentation task.

In particular, Question 2 asked who has the prime responsibility to create and maintain certain types of software documentation. Question 3 asked who has the prime responsibility to verify and validate the information contained in the document types from Question 2. Participants selected the appropriate role for the following types of software documentation: requirements, specifications, detailed design, low level design, architectural, and testing / quality documents

The results to Question 2 and 3 are summarized below in Table 3-11. For further analysis of the documentation responsibilities please refer to Chapter 4, Section 4.1.

Table 3-11: Who has the prime responsibility for the following documentation tasks

Question	Description	Client	Manager	Architect	Developer	Technical Writer	N/A
2 – Create / maintain	Requirements	9	10	6	1	1	0
	Specifications	1	9	13	0	2	2
	Detailed Design	0	1	22	2	0	2
	Low Level Design	0	1	10	10	1	3
	Architectural	0	2	24	1	0	0
	Quality Documents	0	7	5	6	2	6
3 – Verify / validate	Requirements	14	9	3	0	1	0
	Specifications	8	10	4	3	0	2
	Detailed Design	0	5	13	7	0	2
	Low Level Design	1	4	9	9	0	2
	Architectural	1	4	18	4	0	0
	Quality Documents	4	7	2	5	2	5

3.9.5 Documentation Inefficiencies

This section highlights questions that identify potential inefficiencies with undertaking documentation activities. The results indicate the level to which the participants believe that a particular inefficiency is relevant to software projects.

Question 23 asked if the cost of maintaining most software documents outweighs the benefits of having such documents up to date. The premise behind the question is that software projects do benefit from documentation, but at what cost? As outlined in Chapter 1, software engineering is a process of building software now, and preparing to build more software later. The role of documentation is mostly to help in future software development. As such, this question was interested in knowing whether the participants believe the process of documentation is worth the current effort.

Question 32 asked whether the participant would rather refactor / update / debug / test a system than document it in its present (and flawed) state. This question indirectly asked the same question as Question 23. Although there may be many reasons why a participant would prefer to do other activities rather than documentation; the reason is most likely partly due to the fact that participants believe their time would be better spent refactoring, maintaining, debugging or testing the system.

Question 26 asked if software documentation concentrates too heavily on high level issues rather than the important implementation details.

Table 3-12: Document Inefficiencies; higher ratings indicate stronger agreement

Question	Mean	St. Dev	Mode	Strongly Disagree (1)	Somewhat Disagree (2)	Indifferent (3)	Somewhat Agree (4)	Strongly Agree (5)	n/a
23 - maintenance cost outweighs benefits	2.59	1.42	2	13	14	4	8	6	0
32 - rather refactor than document	3.50	1.19	4	2	10	4	18	9	3
26 - too high level	2.39	1.06	2	7	23	5	9	1	1

3.9.6 Documentation Quality Under External Factors

This section outlines the participants' perception of documentation quality under certain external factors and constraints.

Question 7 asked the participants based on their own experience how effective they find the available software documentation for a software project under certain circumstances. The participants rated between (1) as completely useless and (5) as extremely effective.

Table 3-13: Documentation effectiveness under certain conditions

Question 7	Mean	St. Dev	Mode	Completely Useless (1)	Somewhat useless (2)	Indifferent (3)	Software effective (4)	Completely effective (5)	N / A
a – learning a system	3.64	1.13	4	1	4	6	10	7	0
b – maintaining a system	3.04	0.96	3	1	7	9	8	1	2
c – testing a system	3.46	1.17	4	1	6	4	10	5	2
d – others are unavailable	3.38	1.20	4	2	4	7	8	5	2
e – answering to management	2.96	1.19	3	3	7	8	6	3	1
f – big-picture information	3.39	1.31	5	2	6	7	5	8	0
g – in-depth information	2.929	1.215	2	2	11	6	5	4	0
h – new software	3.571	1.034	4	0	5	8	9	6	0
i – mature software	3	1.054	3	2	7	10	7	2	0

Question 8 asked the participants, in their own experience, how well maintained is the documentation in certain types of software projects. The participants rated between (1) as not maintained at all and (5) as extremely well maintained. The results are summarized in Table 3-14.

Table 3-14: How well maintained is documentation in the following types of projects

Question 8	Mean	St. Dev	Mode	Not (1)	Rarely (2)	Somewhat (3)	Well (4)	Extremely well (5)	N / A
a – new / recent	3.39	1.03	3	1	4	10	9	4	0
b – mature without new features	2.96	1.00	3	1	8	13	3	3	0
c – mature with new features	2.39	1.10	2	5	13	6	2	2	0
d – agile / lightweight	2.63	1.07	3	3	5	8	2	1	7
e – open source	3.29	1.38	2	2	6	2	6	5	6

Question 12 asked how relevant are several factors in causing software documentation to be out of sync with the system it describes. The results of question 12 are outlined in Table 3-15.

Understanding what external factors affect the project team's ability to maintain its documentation will help identify priorities among software professionals' daily tasks.

The most relevance factors that contribute to out-dated documentation include time, lack of motivation and the perception that there is little benefit in document maintenance. The least relevant factors are the high costs of documentation maintenance and high staff turnover.

It is interesting to note that most participants believed that individuals do not have the time or motivation to maintain a project's documentation. Similarly, most believe that documentation remains out-dated because individuals see little benefit it its maintenance. Yet, as is illustrated in Section 3.9.7, most participants also believe that having documentation up-to-date is extremely important. For further details, please refer to Chapter 4, Section 4.4.

Table 3-15: External factors influencing documentation quality

Question 12 - Constraints on	Mean	St. Dev	Mode	Completely irrelevant (1)	Somewhat irrelevant (2)	Indifferent (3)	Somewhat relevant (4)	Extremely relevant (5)	N / A
a – Time	4.07	1.15	5	1	2	5	6	14	0
b – Budget	3.39	1.07	3	0	6	11	5	6	0
c – Maintenance costs not worth the effort	3.11	1.37	3	4	6	7	5	6	0
d – Rapid changes in requirements	3.68	1.28	5	2	3	7	6	10	0
e – Rapid staff turnover	3.07	1.36	2	2	11	3	5	6	1
f – Team does not believe in documentation	3.86	1.27	5	2	2	6	6	12	0
g – Team is unmotivated to document	4.04	1.35	5	3	1	3	6	15	0
h – Team sees little benefit in maintenance	4.00	1.22	5	2	1	5	7	13	0

3.9.7 Document Attributes Affecting Document Quality

This section highlights the participants views about what attributes are most important in creating effective software documentation. As well, participants rated how much certain factors contribute to documentation not being updated when project changes occur.

Question 9 asked how important certain documentation attributes are in creating effective software documentation. The lists of attributes used in Question 9 were refined based on the feedback from the pilot study (see Section 3.2). For a complete description of the attributes in Question 9 please refer to Appendix D. The results of Question 9 are outlined in Table 3-16.

Other potentially relevant documentation attributes include consistency, easy of use, clarity of vocabulary, readability, conciseness, correctness and level of detail. Although an investigation

into the relevance of the attributes listed above could provide insight into creating more useful documentation, these attributes were not included in our survey for several reasons. First, the survey length was taken into consideration. As seen in Table 3.7 roughly 54% of all participants answered this question. Adding further attributes would most likely have further reduced the response rate. Second, our list is composed primarily of measurable attributes. Our intentions were to discover if such measurable attributes are relevant. If so, then we can build models based on these relevant attributes and measure their values in hopes of better ranking documentation. Chapter 6 elaborates on modeling documentation relevance.

Table 3-16: How important are the following attributes in creating effective documentation

Question 9	Mean	St. Dev	Mode	Not (1)	Rarely (2)	Somewhat (3)	Very (4)	Extremely (5)	N / A
Content	4.85	0.36	5	4	2	9	10	2	0
a – Up-to-date	4.35	0.69	5	0	1	4	11	11	0
b – Availability	4.19	0.83	4	0	1	10	8	8	0
c – Examples	4.19	0.79	4	1	8	6	7	5	0
d – Organization	3.85	0.91	3	0	6	11	7	3	0
e – Type	3.78	1.05	4	5	9	8	4	0	0
f – Diagrams	3.44	1.09	4	7	6	6	6	2	0
g – Navigation	3.26	1.20	2	0	0	0	4	23	1
h – Structure	3.26	0.94	3	1	2	6	11	7	0
i – Style	3.26	0.94	3	5	8	8	2	3	1
j – Length	3.15	1.17	4	2	4	15	6	0	0
k – Spelling	2.93	0.83	3	1	4	11	9	2	0
l – Author	2.63	1.31	1	0	0	3	11	12	0
m – Influence	2.62	1.24	3	1	5	6	11	4	0
n – Format	2.42	0.99	2	0	1	3	13	10	0

From the data above, the four most important attributes about documentation include its content, maintenance, availability and use of examples. Conversely, the least important attributes include the author, influence from management to use it and documents format (.pdf, .html. doc, etc).

3.9.8 Comparisons to past projects

This section outlines the participants' perceptions comparing current to past projects. Factors such as software quality and documentation quality are examined. As well, we investigate the amount of documentation being produced relative to past projects.

Question 38 asked the participants to compare the amount of documentation being produced, relative to delivered software, between past and present projects. The results of Question 38 are shown below in Table 3-17.

Table 3-17: Relative to past projects, how much documentation is being produced

Question 38	Mean	St. Dev	Mode	Extremely Less (1)	Somewhat less (2)	Same (3)	Somewhat more (4)	Extremely more (5)	n/a
a – Requirements	3.19	1.17	3	3	2	12	5	4	0
b – Specifications	3.08	1.10	3	2	4	11	4	3	2
c – Detailed Design	2.88	1.03	3	3	4	11	5	1	2
d – Low Level Design	2.79	1.02	3	3	5	11	4	1	1
e – Architectural	3.12	0.93	3	2	2	13	7	1	1
f – Testing / Quality Documents	3.33	1.05	3	1	3	11	5	4	2

Question 40 asked the participants to compare the quality of documentation on their current project relative to past one. Table 3-18 highlights the results to Question 40. For a detailed listing of the available options, please refer to Appendix D.

Table 3-18: Relative to past projects, what is the quality of documentation being produced

Question 40	Mean	St. Dev	Mode	Extremely lower (1)	Somewhat lower (2)	Same (3)	Somewhat higher (4)	Extremely higher (5)	N / A
a – Maintenance	3.39	0.94	3	0	4	9	7	3	2
b – Writing Style	3.42	0.93	4	2	0	9	12	1	1
c – Navigation	3.38	0.71	3	0	2	12	9	1	1
d – Searching	3.38	0.88	3	0	3	12	6	3	1
e – Updated	3.25	0.85	3	1	1	15	5	2	1

Question 39 asked the participants to compare the quality of the software being delivered on their current project relative to past ones. Table 3-19 below shows the results of Question 39.

Table 3-19: Relative to past projects, what is the quality of software being produced

Question 39	Mean	St. Dev	Mode	Extremely lower (1)	Somewhat lower (2)	Same (3)	Somewhat higher (4)	Extremely higher (5)	N / A
a – # defects*	3.30	1.06	4	2	2	8	9	2	3
b – Pride	3.28	0.98	3	0	6	9	7	3	1
c – Manager satisfaction	3.16	0.90	3	1	3	14	5	2	1
d – Customer Satisfaction	3.59	0.85	4	0	2	8	9	3	4
e – On-time	3.39	1.12	3	0	6	7	5	5	3
f – on-budget	3.26	1.10	4	1	5	7	7	3	3

* Lower quality number of defects implies more (and more severe) defects relative to past projects

From the results above we see that the amount of documentation being produced is roughly the same as past projects, but the quality has increased somewhat. We also note that overall, software project quality appears to be higher, if not the same, as past projects. Finally, we see the mixed results for a projects ability to be completed on time and on budget.

3.10 Questionnaire Limitations

This section will highlight limitations in our survey that were discovered following the analysis of our work. The items listed below will assist individuals interested in conducting similar research based on the survey presented in this chapter.

The Testing / Quality Assurance role is missing from the survey. Our intentions were that the Testing / QA role was a specialization of a software developer as opposed to being its own unique category. The results of several questions regarding these software roles may have been better represented if this role (and potentially others, such as Configuration / Build engineer) were included in the survey.

The survey ambiguously defined the notion of being up-to-date (and conversely being out-of-date). Our implicit definition was simply the degree to which an artefact reflects the assumed reality of the object to which it is documenting. Being more precise about the attributes of being up-to-date, as well as possibly providing a gradient for the up-to-dateness (such as ranging between being very up-to-date and very out-of-date) may provide greater insight regarding the value of documentation maintenance in future research in this area.

The survey ambiguously defined useful documentation (and conversely not-so-useful documentation). Our implicit definition was the degree to which an artefact helps the reader to complete his / her next task (whether that be to create more documentation, fix a system defect or support additional functionality). Similar to the previous point, a clearer, more explicit definition may help improve future surveys regarding documentation.

In general, some argue that the overall approach to examining software documentation was too broad. We justify our approach by stating that our intentions were purposely broad in scope. We considered all types of documentation and artefacts to search for common themes regarding usefulness and documentation relevance. Future work in this area may wish to draw on the inferences drawn from our work and consider narrowing the scope to further understand the relevance of documentation in particular situations and software scenarios.

3.11 Summary

The data presented from the documentation survey described in this chapter will be used to support the ideas, models and concepts presented throughout this thesis. For a more complete listing of the survey questions and raw results, please refer to Appendix A to Appendix E.

CHAPTER 4: DOCUMENTATION PRIORITIES

This chapter expands on findings from the documentation survey outlined in Chapter 3. Section 3.8 related to the role documentation played in a software project. One of the goals of this survey was to uncover how software documentation is used in industry and the extent to which, and under what circumstances, documentation can be effective. The data suggest there are somewhat conflicting views of the importance of documentation maintenance. In particular, participants responded that not-so-up-to-date documents could still be an effective resource. Conversely, the extent to which a document is up-to-date was selected as one of the most important factors in determining its effectiveness. The results suggest that the software industry and academia may overemphasize the importance of document maintenance relative to a software professional's tolerance of out-dated content.

4.1 Using, Maintaining and Verifying Documentation

In this section, we analyse the results of Question 2, which asked who has the prime responsibility to create and maintain software documentation. Similarly, Question 3 asked for the individual responsible to verify and validate those documents.

Table 4-1 shows the types of individuals most likely to create / maintain different kinds of documentation (results from Question 2) as well those most likely to verify / validate those documents (results from Question 3).

Table 4-1: Documentation maintenance responsibilities (Question 2, 3)

Document Type	Preferred Role to create / maintain document types (% agreement)	Preferred Role to verify / validate document types (% agreement)
Requirements	Manager / Project Leader (37%)	Customer(s) / Client(s) (52%)
Specifications	Software Architects (52%)	Manager / Project Leader (40%)
Detailed Design	Software Architects (88%)	Software Architects (52%)
Low Level Design	Software Developers (90 %)*	Software Developers (78%)*
Architectural	Software Architects (89%)	Software Architects (67%)
Testing / QA	Manager / Project Leader (35%)	Manager / Project Leader (35%)

*This result is an equal combination of Software Architects / Sr. Developers and Jr. Developers

Additional results extracted from Question 2 and 3 are summarized below:

- Design documents (architectural, detailed and low-level) are mostly maintained, verified and validated by senior developers and architects, although junior developers are equally likely to have the same responsibility to maintain, verify and validate low-level documentation.
- A considerable percentage of participants selected 'not applicable' when asked to categorize who maintains (23%) or validates (20%) testing and quality documents. This result may suggests that testing documents are maintained, verified and validated by individuals with software roles not listed in this question, such as QA or Usability specialist.
- Technical writers are rarely employed to maintain any of these types of documents. This finding questions statements in [45] that software engineers should have a limited role in software documentation.
- Managers provided the most consistent results. In general, these individuals identified themselves having the most responsibility regarding the documentation process.
- Managers thought they should validate requirements whereas developers believed clients should do it.
- Requirements (req't) and specification (specs) documents are usually maintained by different people from those who verify and validate them.

Question 6 asked how often the participants consult available software documentation. From Chapter 3, Section 3.9.1 we saw that the results were diverse varying from never to always. Table 4-2 analyzes Question 6 based on the categories outlined in Chapter 3, Section 3.4.

Table 4-2: Extent to which the most consulted document type is typically consulted. A higher consultation value means the document is referenced more often

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

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

4.2 Relevant Document Attributes

This section discusses how certain attributes contribute to a document's effectiveness. The results are taken from Question 9 of the survey, asking how important each item is in helping to create effective software documentation. Please refer to Chapter 3, Section 3.9.7 for the preliminary results of Question 9.

A summary and analysis of the results of Question 9 reveal the following observations:

- Content is the most important factor. All document tasks (creation, maintenance, verification, validation) should always keep the target audience in mind. Effective content is the key to effective documentation.
- Extent to which a document is up-to-date is the second most important factor. Although seemingly intuitive and accepted [[25], [41], and [51]], the following sections will provide a different interpretation of this result.
- A document's format (.pdf, .doc, .html), its author, influence from management to use it and the quality of spelling and grammar have low correlation with the document's effectiveness.

Relating to documentation engineering in the large, and based on the data outlined above, documentation technologies should strive to facilitate the above qualities that promote a document's usefulness. In particular, our data suggest that technologies should:

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

4.3 Document Maintenance?

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

Question 4 asked how long it takes for supporting documentation to be updated to reflect changes made to other areas of the software system.

Table 4-3 illustrates the preferred score (mode), the percentage of responses, as well as its semantics of the preferred score. Please refer to Chapter 3, Section 3.9.1 for a complete listing of the results to Question 4.

Table 4-3: How often is documentation updated? A higher the score means a document is more quickly updated following changes to the system

Document Type	Mode	% of Mode	Semantic Meaning of Score
Requirements	2	52	Rarely updated
Specifications	2	46	Rarely updated
Detailed Design	2	42	Rarely updated
Low Level Design	2	50	Rarely updated
Architectural	2	40	Rarely updated
Testing / Quality Documents	5	41	Updated within days of a change

The results above might seem trivial in that most already believe that software documentation is being rarely updated. Nonetheless, our results help provide substance to their claims and indeed our results do suggest that documentation is rarely maintained.

Based on the participant categorization (see Chapter 3, Section 3.4), several statistically relevant differences occur between agile and conventional participants, as well as those grouped as iterative and waterfall.

Table 4-4 summarizes the differences between agile and conventional individuals with respect to document maintenance, and Table 4-5 between iterative and waterfall individuals.

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

Table 4-4: Documentation Update Time (Agile vs. Conventional); a higher the score means a document is more quickly updated following changes to the system

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

* Statistically significant differences between the means with 95% confidence

Table 4-5: Documentation Update Time (Iterative vs. Waterfall), a higher the score means a document is more quickly updated following changes to the system

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

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

Question 20 asked to what degree the participants agreed that documentation is always out of date. The following observations compare the results of agile and conventional participants (as described in Chapter 3, Section 3.4).

As noted in Chapter 3, Section 3.9.1, the data ranged from 'strongly disagree' to 'strongly agree'. Many participants (43%) somewhat agreed with that statement, but a considerable number (25%) also strongly agreed. In particular, the agile participants were statistically more likely to agree

with this statement (mean of 3.83, st. dev 1.20) compared to the conventional participants (mean of 3.08, st. dev 1.29). There is also suggestive, but not statistically significant evidence that iterative participants are also more likely to agree that documentation is always out of date as compared to the waterfall participants. These results help affirm the results shown in Table 4-4 and Table 4-5 that iterative and agile participants are less likely to update documentation. As well, the data supports the general consensus that documentation is rarely updated.

The evidence that agile and iterative individuals work in projects where documentation is less frequently updated and almost always outdated does not imply that these projects are of lower quality or that proper software engineering practices are not in place.

Software project quality may be high despite a lack of documentation maintenance, and, as many have suggested (such as [4], [25]), inferring a lack in documentation quality. One might argue that since a software engineering's primary role is to deliver software (with its secondary role to facilitate future software development) then it may be that the role of documentation, based on current practices in software engineering, is of little importance with respect to software quality. Unfortunately our data suggests otherwise, as most individuals in our survey believe in the importance of documentation as well as the fact that the documentation available to them is important (see Chapter 3, Section 3.9.1). As well, most participants somewhat agree that both software project quality and documentation quality are improving (see Chapter 3, Section 3.9.8).

It seems reasonable to believe there should be a high correlation between a software project's success and the quality of the documentation available. However, we also believe that many individuals may have an inappropriate parsing pattern (explained by Cockburn in [15], p. 3 and paraphrased as the techniques that an individual uses to processes his or her surroundings) to identify documentation quality. Instead of investigating the correlation between software quality and documentation quality, we will investigate the relationship between documentation quality and the extent to which a document is up-to-date. Our reason for taking this approach will become clearer in the following section.

4.4 The Up-to-date Double Standard

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

Question 21 of the survey asked participants if they believe that documentation can be useful even though it is not always up-to-date.

Table 4-6 outlines the results of Question 21 based on the categories outlined in Chapter 3, Section 3.4.

Table 4-6: Can out-dated documentation be useful? The higher the score, the more the participant agrees that out-dated documentation can be useful.

Participant Category	Mean	St. Dev.	Percentage that Strongly Agree
All	4.0	0.98	28 %
Waterfall*	4.1	0.75	38 %
Iterative*	3.5	0.44	15 %
Agile	3.9	0.74	28 %
Conventional	4.1	0.87	29 %
Manager*	3.3	0.76	8 %
Developer*	4.1	0.67	35

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

There is overwhelming agreement that out-dated documentation is still quite useful. In all but two categories the mean was at or above 4.0 (the average participant somewhat agrees with the question statement). This observation questions both common intuition as well as past statements that documentation is practically useless unless accurate and kept up to date (for instance [3], [23], and [41]).

The conflict might be the assumed association between being up to date and correctness, and inversely not-so-up-to-date with incorrect. Some sources argue that being out-dated implies the information is incorrect and thus not reliable. This unreliability then affects the document's credibility and hence its effectiveness [41].

The above reasoning is based on the notion that documentation must present facts, and facts are only useful when they are accurate and up to date. Conversely, some argue that the purpose of documentation is to convey knowledge or information [15]. The system's source code is the artefact that presents the facts, where as the supporting documents facilitate higher-level views of those facts. A document that instils knowledge in its audience can then be deemed effective, somewhat regardless of its age and the extent to which it is up-to-date [15].

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

Extent to which a document is up-to-date was one of the document attributes listed in Question 9. To recall, Question 9 asked to what degree the following attributes contribute to effective software documentation.

Table 4-7 outlines the mean, standard deviation and the percentage of participants who rated this item a 5 (the item is one of the *most* important factors to determine a documents importance).

Table 4-7: The importance of up-to-date documents; the higher the rating the more important the attribute of being up-to-date.

Participant Category	Mean	St. Dev.	Percentage of participants rating 'up-to-date' as one of the most important attributes
All	4.3	0.89	46 %
Waterfall	4.4	1.25	50 %
Iterative	4.5	3.33	50 %
Agile	4.3	0.73	44 %
Conventional	4.3	1.45	43 %
Manager	4.0	2.23	20 %
Developer	4.4	1.14	56 %

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

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

This disagreement that out-dated documentation can be useful, but that being up-to-date is a crucial element of useful documentation may influence individuals to over-estimate the importance of the up-datedness of a document relative to several other factors including content, availability and use of examples. Although we can conclude that it would be very nice if our

documents were up to date, we should not necessarily consider them to be of lower quality solely because they are out-dated. More importantly, it seems wasteful to attempt to consistently and regularly assure that all documents are up-dated for the mere sake of keeping the documents current [15]. Software documentation should evolve with the project team based on the needs and available resources of those team members. Maintenance for its own sake detracts software professionals for other potentially more important activities including software construction.

A reviewer of a paper based on this chapter arrived at a seemingly logical conclusion based on the information presented in this chapter. To avoid further readers from drawing similar but unwarranted conclusions, we will address the issue directly.

An anonymous reviewer commented

> "[T]he survey is based on current practices, which is a dangerous starting point for asking what improvements could be made: 'Our documentation is sloppy and out of date, and we don't use it well, so we will improve our process by spending even less effort on documentation' may or may not be a valid argument."

It is important to understand that our findings, based on the data from the survey as well as from the existing literature, do not promote improving the documentation process by merely ignoring it. Rather, our goal is to help software project team members realize that out-dated content can still be useful and that software teams should analyse their own documentation needs to help produce documentation that is better suited and integrated in the development process. Managers and project leaders can approach the document situation from a slightly more abstract level to identify the future needs of the team members and coordinate sufficient documentation without hindering current development.

In general, our recommendation is that the act of documentation should be scrutinized in the same manner that new features are added to a system; with care.

Our primary argument is that documentation should not be updated *merely* because it is outdated. Instead, documentation should be updated when a real, not perceived, benefit will be achieved by maintaining the document. The survey data showed that individuals believe that out-dated documentation can still be useful, and therefore being out-dated is not always a valid argument for documentation maintenance. A better rationale for documentation maintenance is the combination of the situation where the team requires information, but the current selection of

artefacts no longer effectively conveys that information. Information needs change throughout the lifespan of a software project and the fact that an artefact may no longer convey information is also usually insufficient to warrant maintenance. Individuals must also need and use the artefact's information.

4.5 Agile vs. Conventional Perceptions of Documentation

This section describes in more detail how the agile and conventional participants (as described in Chapter 3, Section 3.4) use and perceive documentation in practice. Table 4-8 compares the results of the Questions 14, 15, 16, 17 and 34 between agile and conventional participants.

Table 4-8: Perception of Documentation (Agile Vs. Conventional); a higher score means a higher agreement to the statement

Question	Agile		Conventional	
	Mean	St. Dev	Mean	St. Dev
14 - not useful in our organization	2.35	1.40	2.41	1.54
15** - easy to understand and navigate	3.56	1.19	3.06	1.39
16* - brief and to the point	3.56	1.04	2.94	1.03
17 - easy to locate	2.80	1.12	2.82	1.13
34* - too many documents available	2.88	1.19	3.41	1.33

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

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

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

Despite observations that documentation is rarely updated (see Section 4.3), agile participants feel that the documentation they reference is brief, to the point and easy to navigate; more so than conventional participants. Also, a considerable number of agile participants strongly agree (39%)

and somewhat agree (22%) that the documentation in their projects is useful. Finally, agile participants indicated that they were less likely to maintain documentation relative to conventional participants (see Section 4.3). Combined, these results support our hypothesis stated earlier that present mental parsing patterns for software documentation may not be the most appropriate (see Section 4.3). That is, being up-to-date may not be the most appropriate metric to consider when analysing the relevance of software documentation.

4.6 Software Project Quality

This section comments on the participants' comparison of current software project relative to past projects with respect to overall quality, as well as the quality of the software documentation. Please refer to Chapter 3, Section 3.9.8 for a more detailed listing of the results discussed below.

Most participants cited small improvements to software quality based on decreased defects, increased customer satisfaction and to a lesser extent improved project delivery (both with respect to time and budget). Similar improvements to software documentation were also noted. Combined with the observations that documentation is rarely maintained (see Section 4.3), the results suggests the improvement in documentation quality is due to factors beyond maintenance; such as those attributes discussed in Section 4.2. It is also important to note that most participants believed the amount of documentation produced in a project has stayed roughly the same compared to past projects; suggesting that the 'document everything' [25] approach is not a necessity for quality documentation.

If one believes that useful documentation is highly correlated to project quality, as we do, then one can infer low correlation between a document's quality (and hence usefulness) and the extent to which a document is up-to-date.

4.7 Summary

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

- Document content can be relevant, even if it is not up to date. (However, keeping it up to date is still a good objective).
- Documentation is better described as a tool for communication as opposed to simply a description of fact regarding a software system's source code.

The conclusions cited above will help decision makers choose more appropriate documentation strategies and technologies based on needs as opposed to generic expectations.

Once we can admit that documentation is often out-dated and inconsistent, we can then appreciate and utilize it appropriately as a tool of communication. This tool can then be judged based on its ability to communicate and instil knowledge in the reader as opposed to merely presenting facts about a system's source code and its structure.

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

CHAPTER 5: THE RELEVANCE OF SOFTWARE DOCUMENTATION TOOLS AND TECHNOLOGIES

This chapter highlights the results from our survey from a tools and technologies perspective. One of the goals of this survey was to uncover the perceived relevance (or lack thereof) of software documentation, and the tools and technologies used to maintain, verify and validate such documents. The survey results highlight the preferences towards and aversions against software documentation tools. Participants agree that documentation tools should seek to better extract knowledge from core resources. These resources include the system's source code, test code and changes that occur to both. Resulting technologies could then help reduce the effort required for documentation maintenance, something that is shown to rarely occur. The data report compelling evidence that software professionals value tools that improve automation of the documentation process, as well as technologies facilitating its maintenance.

5.1 Software Technologies in Practice

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

Question 36 asked,

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

Table 5-1outlines the most frequently cited technologies based on 41 participants that answered Question 36.

Table 5-1: Useful documentation technologies

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

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

Question 37 asked,

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

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

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

- Word and text processors. Although perhaps not the most efficient means of communication, these processors are flexible and in general easy to use.
- Automated documentation tools. These tools improve document maintenance by practically removing the need for it. Maintaining automatically created documents should be as simple and straightforward as building the software project.

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

> "The purpose of documentation is communication. Some tools are overapplied and the communication factor is lost. For example, a low level design tool should

be easy to use in a brainstorming type of scenario when developers are hashing out the way to do something (currently, whiteboards are very effective for such interactions). If a low level design tool thinks its artifacts are an essential part of the software documentation to be maintained rigorously beyond that collaboration session, then that tool has an unwelcome fault."

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

5.2 Evolving Documentation Needs

This section affirms the fact that the information needs of software professionals evolve over the lifetime of a project [[1], [15]].

Question 27 asked to what extent participants agreed that

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

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

Overall, many participants strongly (32%) and somewhat (46%) agreed that their needs for different types of documentation change throughout a project's lifecycle. Only a small portion of participants strongly disagreed (7%) with the statement. The results are consistent among all participant categories outlined in Chapter 3 (Section 3.4).

Question 22 asked to what degree the participant agreed that

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

Table 5-2 compares the percentage of individuals that disagreed (score of 1 or 2) with those that agreed (score of 4 or 5) with Question 22 based on the categories outlined in Chapter 3, Section 3.4.

Table 5-2: Does documentation have a finite lifetime?

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

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

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

As mentioned, from Question 27 we see that the document needs of most participants change over time. This result suggests that these individuals would also agree that a document has a finite useful lifetime. If this is true, then the disagreement in Question 22 most likely stems from how these documents should be treated once they are no longer used, either archived or discarded. The fear to discard documents may be the result of not knowing if, how, or when others might use a particular document. There may also be a reluctance to discard documentation due to the effort and resources required to produce it. It has been shown that archiving all documents and related artefacts has proved to be unsuccessful [25] – it tends to be impossible to search and use such a collection.

If few resources were expended to produce a document, then perhaps individuals would be less hesitant to discard such documentation. As well, if archiving documentation could be based on usage and its management somewhat automated, then improved navigation would be possible, improving efforts to retrieve pertinent and hence better documentation [33]. In later sections, we will provide additional insight into a document's useful lifetime.

5.3 Support for Documentation Automation

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

Question 24 asked to what extent you agree that

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

Question 30 asked to what extent you agree that

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

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

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

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

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

5.4 Need for Documentation Tracking

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

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

Question 31 asked to what extent you agree that

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

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

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

First, the technology should complement existing documentation tools as well as attempt to seamlessly integrate itself into a project. From Section 5.1, we see that a variety of tools are employed with respect to documentation and it is unlikely that individuals will adopt new technologies if it means they must abandon current ones. As well, this technology should not focus on documentation; but rather, on tracking changes between documents and source code. Finally, all team members should have access to, as well as the ability to influence the information gathered by the technology.

Second, the technology must be able to relate documents to source code, as well as to other documents. When changes occur to source code, we cannot predict where changes may be required in the documentation unless appropriate relationships exist between the two. To support such relationships, documentation technologies should learn from the concept of *authoritativeness* in hyperlinked environments [33]. This concept suggests that documents containing the highest number of outgoing references, or are referenced the most, are the ones

most likely to be worthy of attention. In addition, it is important that users be able to easily input their own relationships among documents, as well as to remove any automatically generated ones that appear irrelevant. This manual intervention helps to improve the quality of the relationships between documents and source code.

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

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

- First, possible inconsistencies between documents and source code are highlighted for the user, helping to provide a maintenance map when updating documentation.
- Second, maintenance can be prioritized based on user feedback as opposed to by assumption. Discrepancies will be highlighted in two main ways: automatically based on changes to the software or manually through user intervention. As complete document maintenance is impractical, maintainers can first concentrate on user feedback. Based on the notion that out-dated documentation can still be a useful (see Chapter 4, Section 4.4 and 4.5), one can then base maintenance on feedback regarding document inconsistencies as opposed to solely on the fact that a document may be outdated.

5.5 Supporting Lightweight Documents

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

Key features of lightweight technologies include supporting

- Content creation over maintenance. Documents are rarely maintained. Technologies should be easy to use and should make it easy for the author to capture knowledge with; they should support the creation of information and not necessarily its maintenance.

- Ideas over accuracy. Chapter 4 concluded that documentation best serves as a communication medium as opposed to a fact base that must be precise and up-to-date. As such, tools should facilitate the communication of ideas, sharing with others, and prompt feedback. For example, the use of whiteboards in combination with digital cameras or printing capabilities has been known to be an effective means to create documentation [15].

- Simplified features. The tools most preferred for documentation include word processors and text editors (see Section 5.1). These technologies provide extensive freedom to users and should be a role model for most lightweight documentation tools.

- Automated archiving. As seen in Section 5.2, many individuals are unlikely to discard documentation. However, too much documentation can decrease the usefulness of the entire repository [25]. As such, it would be beneficial to have lightweight technologies archive such documents based on user preferences and document usage.

- Reader feedback. Feedback is an important tool of communication [[1], [15]]. As a consequence, the easier the reader can provide feedback, the more attuned the writer can become at providing useful content.

5.6 Summary

Based on the April 2002 survey from the perspective of documentation tools and technologies, the following observations can be made. In particular, documentation technologies should

- Strive to be easy to use, lightweight in nature and support disposable solutions.

- Enable writers to quickly and efficiently communicate ideas. The ability to communicate is often more useful than providing tools that support strict validation and verification rules for building software facts about a system.

- Enable readers to quickly and efficiently provide feedback to the writers. This feedback mechanism will allow writers to better attend to the needs of its audience, as well as validate the need for a particular document.

As we learn more about how technology can improve documentation, we hope for improved techniques to communicate information about a software system in a clear and concise manner.

CHAPTER 6: DOCUMENTATION RELEVANCE MODELING: DOCUMENT AURA

Documentation comprises several measurable attributes, some of which are better able to predict document relevance. Based on the work presented earlier in this thesis, and combined with existing research in the field of information retrieval and buffer management, this chapter introduces what we call document Aura,

> *a prediction of an artefact's relevance in a particular situation based on its usefulness, referential decay and authority.*

We use this as a measure to rank documentation based on the environment in which an artefact exists as well the conditions under which the information is requested.

6.1 Documentation Metrics

Our approach to ranking documentation is based on three components; usefulness, referential decay and authority. Each component contributes to Document Aura in several ways and under various conditions. As well, each component is comprised of one or more parameters that are used in its computation. The following subsections will elaborate on the individual metrics used to calculate document Aura.

6.1.1 Usefulness as a Measure of Recency, Frequency and Feedback

The usefulness of an artefact is

> *a measure of how well a document's content can provide knowledge, information and / or insight to its reader with respect to other available artefact consider both the activity and value of the artefact.*

To measure the activity (i.e. its use) of an artefact, we refer back to Lee et al's [34] Least Recently Frequently Used (LRFU) approach to buffer replacement discussed in Chapter 2, Section 2.3.2. To integrate the value of an artefact, we have integrated a feedback mechanism

where the audience of an artefact is able to rate the value of an artefact's use. This section will elaborate on applying the LRFU algorithm, as well as adapting it to better reflect usefulness, not necessarily just use.

To re-state, LRFU uses the frequency with which an item is accessed and adjusts each access based on the recency with which it was made. Higher CRF values indicate a higher likelihood that a document is relevant. LRFU was chosen for its ability to reduce the effects of locality without limiting the timeline of an artefact's history. The algorithm can also be adjusted to reflect any combination of recency versus frequency weighting.

The CRF metric will be used to predict the usefulness of a document. The approach of using both the frequency and recency of use is appropriate to predict future uses (and hence usefulness) of an artefact. As a project progresses, the information needs of the software team change and therefore a measure based purely on frequency would not accurately reflect potential future use. Considering only the recency is also flawed in that it is too short-sighted to provide an accurate gauge of potential future use.

LRFU calculations are concerned only with access of an artefact for the purpose of using it. Valid types of access include browsing an artefact's content or seeking particular information in an artefact, an essential part of program comprehension [46]. In this metric, we have purposely excluded accesses to an artefact that involve changing it. Although keeping a document up-to-date does suggest usefulness, we will leave that to other metrics discussed below. By not considering maintenance in CRF we have a metric that avoids measuring documentation maintenance activities done *for the sake document maintenance*.

We can measure usefulness, beyond merely measuring the amount of use that occurs by allowing the user of a document to provide feedback to rate an artefact's quality based on its intended goal. This approach to rating usage can reduce the effect of inappropriately high ranks achieved by documents that happen to be accessed frequently (perhaps because they are easy to find), but are then found by users to be not useful.

To incorporate feedback into the usefulness of a document, we introduce the notion of artefact value,

> *as an ordinal scale* $V \in \{V_1 = 1, V_2 = 2, V_3 = 3, V_4 = 4\}$ *that states to what degree an artefact offers helpful information and / or guidance.*

The values above correspond as follows; poor quality is 1, inadequate quality is 2, good quality is 3 and excellent quality is 4. The vector, $V(d)$, represents all artefact values of a document d (each one associated with an access to that document).

To determine a feasible measure that fairly represents the notion of usefulness as a measure of activity and value, we first consider the relationship between and relative-weighting factors of the individual metrics. Figure 6-1 below outlines an intuitive ranking of usefulness based on the four primary quadrants. The lower left quadrant is the least useful (value of 1) and the upper right quadrant is the most useful (value of 4). Based on the ranking shown below it is reasonable to combine activity and value using an additive, as opposed to multiplicative, relationship. As the activity of an artefact approaches 0, the overall usefulness does not necessarily also approach 0; a basic prerequisite to multiplicative relationships.

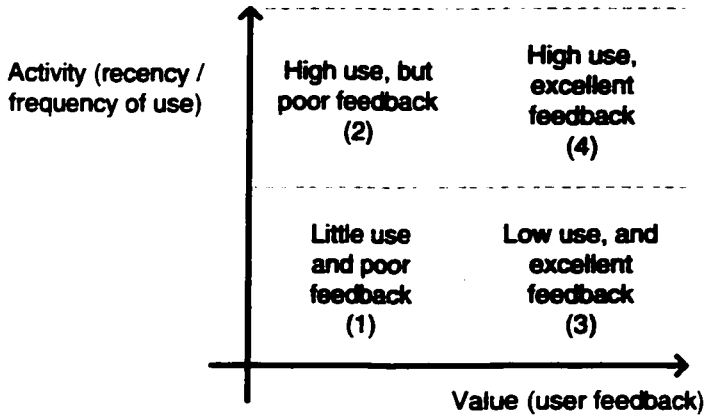

Figure 6-1: The visual ranking of usefulness based on activity and value

A general approach to combining the two measures above to calculate the usefulness, U, of an artefact d at time t_k is shown below in Equation: 6-1

$$U_{t_k}(d) = a * Value + b * Activity$$

Equation: 6-1
Generic Usefulness Metric

Using the value-rating scale described above to measure the artefact value with Lee et al's [33] work regarding LRFU (see Chapter 2, Section 2.3.2) to measure its activity we can arrange the likely order of future accesses based both on perceived quality of an artefact and the recency / frequency with which it has been accessed.

The resulting usefulness equation, labelled $U_{t_k}(d) \in \{R \mid R \geq 1 \wedge R < 5\}$ is shown below in Equation: 6-2.

$$U_{t_k}(d) = MV_{t_k}(d) + Percentile(CRF_{t_{k-1}}(d)) \qquad \text{Equation: 6-2 Usefulness Value}$$

$$MV_{t_k}(d) = V \mid \sum_{i=1}^{4} CRF_{t_k}(V) \geq CRF[V_i(d)] \qquad \text{Equation: 6-3 Most-likely Value}$$

$MV_{t_k}(d)$ is the most-likely artefact value and its meaning is somewhat similar to that of mode. The most-likely metric selects the artefact value of a document d such that its CRF value is a maximum (and hence is most likely to represent the artefact's current value). We avoided using Mode due to the complications of potential past ratings interfering with the affect of current ones (for example, past high past ratings should not out-weigh current low ratings). Using the CRF value we still consider all past accesses, but place more emphasis on current ones. The result is the selection of the most likely artefact value of the next access to the document d.

$Percentile(CRF_{t_{k-1}}(d)) \in \{R \mid R \geq 0 \wedge R < 1\}$ is computed as the percentile, but represented as a fraction, of the LRFU metric (CRF) of accesses to an artefact with respect to all artefacts.

We have somewhat arbitrarily, but also carefully, chosen a weighting factor relating value to activity as 3:1. First, we have avoided the complication in combining an ordinal scale [$MV_{t_k}(d)$] with the rational scale [$CRF_{t_k}(d)$] (as discussed in [11] by Lionel Briand et al). Second, we are using the percentile of $CRF_{t_k}(d)$ as a fraction; therefore the addition of the whole number $MV_{t_k}(d)$ can be accomplished without losing, or distorting the meaning or either metric. The resulting usefulness measure $U_{t_k}(d)$ captures the value of an artefact as a whole number, and the relative percentage of its activity (that is, its recency and frequency of use) as a fraction. By using (and maintaining separate) both the fractional and whole based portions of the metric, we can efficiently capture both the perceived value of the artefact and it's relative recency and frequency of use. Following further empirical studies with regards to the usefulness of an artefact, a more empirically conclusive parameterization of Equation: 6-1 will be possible. From an intuitive perspective, our weighting approach is reasonable based on the data available. Equation: 6-2 also

has the welcome side effect of combining both the value and the activity of an artefact without losing their distorting their original meaning.

In words, the interpretation of Equation: 6-2 is the usefulness of an artefact is first determined by its overall perceived relevance and then rated based on the recency and frequency with which it was accessed. Because the resulting measure $U_{t_k}(d)$ captures both the value of an artefact and its recency and frequency of use, the metric can be used in an assortment of ways and described in Section 6.2.

LRFU/CRF in combination with feedback techniques can provide an efficient means to predict an artefact's usefulness. Its ability to consider all past transactions (i.e. accesses to a document) as well as places emphasis on more current activity makes it suitable for the measure of an artefact's usefulness. Buffer management algorithms have been successfully used in database management systems for some time, as well as more recently in the field of information retrieval and should prove to be relevant in the realm of documentation management.

6.1.2 Approximating Referential Decay

Referential decay is

> *a measure that reflects the relative-likelihood of an artefact containing inconsistencies with other related artefacts, as well as the likelihood of being used in inconsistent ways in related artefacts.*

This is another metric based on the history of an artefact's use (in particular, we are concerned about when an artefact was last updated). With referential decay, we are looking to predict how inconsistent an artefact *may* be, based on measurable attributes of a software system.

Our measure to predict inconsistencies between related artefacts starts by using the artefacts' dates of last modification to obtain hints as to the possible integrity of the artefacts. The higher the *difference* between last modification dates of related artefacts, the higher the probability that these documents contain relative inconsistencies. I.e. in cases where a recently updated artefact references an older one, the older artefact is more likely to contain out-of-date content, resulting in the recent artefact referencing potentially inaccurate information. If the older document had been updated around the same time as the recently updated one, we would have had greater

confidence that its content is up to date. Similarly, in cases where an older artefact references a newer one, the older artefact is more likely to reference content that has since changed in the newer artefact, again contributing to inconsistencies among documents.

Our approach to predicting referential decay *propagates* itself throughout the entire collection of documentation in a manner similar to that of Page Rank discussed in Chapter 2, Section 2.3.3. The logic for propagating referential decay is that referencing an item (or being referenced by an item) with a high decay value suggests that the item itself should also have a high referential decay. Similarly, referencing an artefact with low decay suggests that item as well should have low referential decay. Although an association between two artefacts may be directed, for the purposes of referential decay, the resulting graph used to calculate referential decay is considered bi-directional.

Equation: 6-4 defines the referential decay $RD(A)$ of an artefact A, where artefacts, $T_1,...,T_n$ reference (or are referenced by) A. Equation: 6-5 defines the average weighted age, $W(A)$.

$$RD(A) = (1-d) + d\left(\frac{RD(T_1)}{\overline{W}(T_1)} + ... + \frac{RD(T_n)}{\overline{W}(T_n)}\right)$$

Equation: 6-4
Referential Decay

$$\overline{W}(A) = \frac{\sum_{i=1}^{n} \left| last\_update(T_i) - last\_update(A) \right|}{n}$$

Equation: 6-5
Average Referential Weight

To maintain the referential decay as a probability distribution, a damping factor, $d \in \{0 \le d \le 1\}$, set to 0.85 is used. Our choice of damping factor is the same as that chosen by Brin and Page to calculate Page Rank [12].

To illustrate the application of referential decay, we will consider an example set of artefacts (and their relationships among one another) is shown below in Figure 6-2.

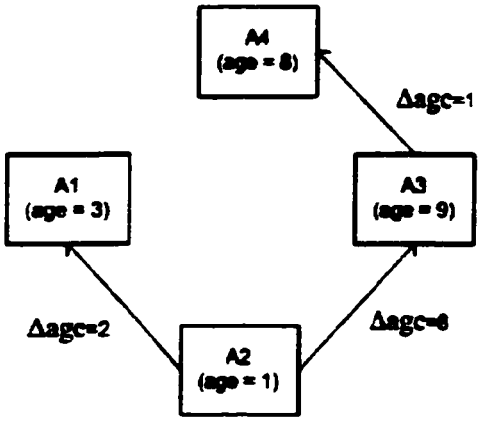

Figure 6-2: An example set of documents, and their relationship among one another

Using the set of documents from Figure 6-2, the referential decay for $\{A1, A2, A3, A4\}$ would be as follows:

Initial RI calculation: $\{A1, A2, A3, A4\} = \{1,1,1,1\}$

First Iteration:

$$A1 = 0.15 + 0.85 \times \left(\frac{1}{\left(\frac{2+8}{2} \right)} \right) = 0.320$$

$$A2 = 0.15 + 0.85 \times \left(\frac{1}{\left(\frac{2}{1} \right)} + \frac{1}{\left(\frac{1+8}{2} \right)} \right) = 0.764$$

$$A3 = 0.15 + 0.85 \times \left(\frac{1}{\left(\frac{2+8}{2} \right)} + \frac{1}{\left(\frac{1}{1} \right)} \right) = 1.170$$

$$A4 = 0.15 + 0.85 \times \left(\frac{1}{\left(\frac{1+8}{2} \right)} \right) = 0.320$$

After 20 Iterations: $\{A1, A2, A3, A4\} = \{0.203, 0.309, 0.386, 0.216\}$

After 20 iterations the results have stabilized and the following observations can be made.

- Even though $A2$ is the most recently updated, it does not have the lowest referential decay, due to the fact that it references $A2$ which has the highest referential decay value.

- The fact that $A4$ was updated around the same time as $A3$ explains why it has the second lowest referential decay value

- $A1$ has the lowest referential decay value because it has been recently updated and it references artefacts with relatively low referential decay value.

A limitation to our approach is that it assumes that all maintenance tasks apply equally over the entire artefact. One could image a situation where only a small section of an artefact is updated To reduce the effects of localized maintenance, the scope of an artefact must be reduced. As an example, consider a document that was originally considered as one artefact decomposing into smaller, more manageable chunks which are themselves artefacts. It is then reasonable that these smaller artefacts could indeed be wholly maintained.

It should be noted that we have reserved the empirical justification of the details of referential decay for future work. It will not be addressed directly in our work. Our justification for incorporating referential decay in document Aura is based, in part, on the results of our survey that being up-to-date is an important aspect of document. Using a measure that predicts the likelihood that an artefact contains inconsistencies, we can better distinguish documents that are likely to contain both accurate and up-to-date information.

Referential decay is not a measure of age, but rather a measure of likelihood that an artefact contains inconsistencies. Therefore an approach, such as ours, that incorporates the maintenance history of an artefact, as well as its relationship among other artefacts, should be a relevant measure to consider when rating the appropriateness of software documentation.

6.1.3 Authoritative Ranking

The application of authority in the context of documentation is similar to the original study of hypertext searching in stand-alone environments. Our approach uses Kleinberg's authoritative ranking technique [33], and is described in detail in Chapter 2, Section 2.3.3. Although present techniques of page ranking and authority are more closely related to the Web, the study of link structures and their relevance in stand-alone environments is still applicable.

An authoritative technique, versus page rank for example, is appropriate for software documentation as the approach actually tracks two separate metrics. First, the approach tracks the

extent to which a document provides an authoritative source of information. Second, the approach also tracks the extent to which a document indexes authoritative sources, and is hence known as a hub. Therefore by using authority, we are able to predict which documents contain the most authoritative content as well the documents that act as hubs for the most authoritative content.

6.2 Applications of Document Aura

Based on the analysis of the results from our survey of software professionals, the following section hypothesizes about possible application of document Aura to three main software engineering scenarios. These scenarios include,

- Using documentation to learn a software system.
- Using documentation to enhance an existing system.
- Maintaining software documentation.

Our investigation of the hypotheses discussed in this section are analyzed further in Chapter 8. Please note, that Hypothesis 1 to 3 implicitly rank documents based on maximized authority, minimized referential decay and maximized usefulness. The focus of Hypotheses 1 to 3 is to determine the relative weighting that each component should have in calculating the overall document Aura under certain conditions.

6.2.1 Learning a Software System

When referring to learning a software system, our expectations from the documentation are high.

As Cockburn's [15] describes in his analysis of levels of behaviour when learning something, individuals "measure success by (a) whether the procedure works and (b) how well they can carry out the procedure." [15] In the context of software documentation, the criteria above suggest the information should be accurate, as well as complete. The above analysis forms the basis for the following hypothesis.

Hypothesis 1: When learning a software system, document Aura calculations should place high emphasis on authority, medium emphasis on referential decay and low emphasis on usefulness.

Hypothesis 1 is summarized below in Table 6-1.

Table 6-1: Document Aura ranking when learning a software system (Hypothesis 1)

Document Aura Component	Component Value	Relative Weight in Aura Calculation
Usefulness:	Maximized	Low
Referential Decay:	Minimized	Medium
Authority:	Maximized	High

6.2.2 Maintaining a Software System

Most of a software engineer's time is spent in maintenance, during which time searching is the most prevalent activity [47]. Our work considers two distinct types of searching which are described below. These types of searching are adaptations of Janet Walker's [54] analysis of how readers access a software manual.

Browsing is

> *searching an artefact (or several artefacts) with the intent to learn overall concepts and not necessarily any one specific bit of information.*

whereas, seeking is

> *searching an artefact (or several artefacts) with the intent to learn / reference some particular information.*

Hypothesis 2: When browsing a software system during maintenance, document Aura calculations should place high emphasis on authority, medium emphasis on usefulness and low emphasis on referential integrity.

Hypothesis 2 is summarized below in Table 6-2.

Table 6-2: Document Aura ranking when browsing a system during maintenance (Hypothesis 2)

Document Aura Component	Component Value	Relative Weight in Aura Calculation
Usefulness:	Maximized	Medium
Referential Decay:	Minimized	Low
Authority:	Maximized	High

Our hypothesis is based on concept of document sufficiency. As our survey results suggested in Chapter 4, an artefact may contain several inconsistencies with regard to the system, but still be useful. Chapter 3, revealed that content is the most important attribute of effective documentation and explains the high importance on authority. As such, artefacts with high authority are those artefacts with the potential for providing the most insight into a particular subject area. We also note that frequency of use is not necessarily an important attribute when browsing a system as these documents may provide abstract knowledge, but not necessarily practical insight that is used on a frequent basis.

A third hypothesis we propose is that when seeking information from the system, document Aura should emphasize high authority, high usefulness, and low referential decay.

Hypothesis 3: When seeking specific information about a software system during maintenance, document Aura calculations should place equally high emphasis on authority, usefulness and referential decay.

Hypothesis 3 is summarized below in Table 6-3.

Table 6-3: Aura ranking when seeking a system during maintenance (Hypothesis 3)

Document Aura Component	Component Value	Relative Weight in Aura Calculation
Usefulness:	Maximized	High
Referential Decay:	Minimized	High
Authority:	Maximized	High

Our justification for the hypothesis above is based on the idea that when you are seeking particular information, it is more harmful than beneficial to find appropriate documentation that is incorrect or misleading. Documentation need not be up-to-date to be useful. However, when seeking answers to a specific question (or questions) documentation that provides incorrect

insight could potentially have adverse side effects. These side effects include introducing defects into the system, or abusing the available software architecture.

Figure 6-3 outlines the relative importance of the three Document Aura factors when learning a system, browsing it, or seeking information. The weighting ratios follow the hypotheses introduced in this section.

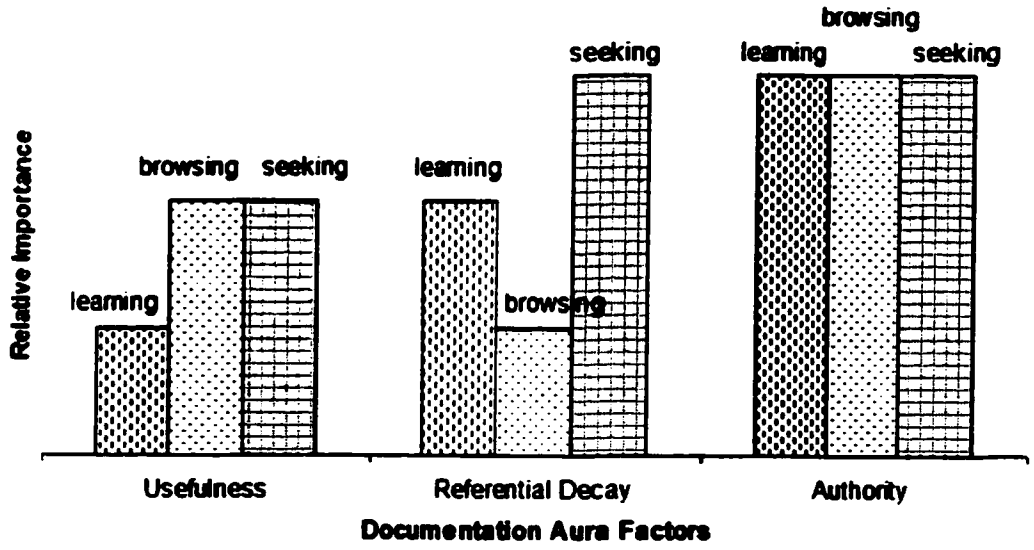

Figure 6-3: Relative importance of document Aura factors when learning a software system, browsing it or seeking information (visualization of Hypothesis 1, 2, and 3)

6.2.3 Prioritizing Document Maintenance

We define documentation maintenance as

the process of refactoring a document's content to improve its relevance.

Document Aura should improve maintenance overall, as well as prioritizing the maintenance effort.

Document Aura opens the line of communication between the writers and their audiences. This communication is both explicit and implicit. Explicit communication will be achieved as the audience is able to provide feedback when using a particular artefact. Based on this feedback,

writers improve overall document relevance by addressing the prevalent needs of its audience. Implicitly, authors will be able to gauge relevance by observing the usage of their artefacts, perceived authority and referential decay.

Implicit communication will be gauged by the factors that influence document Aura. The writers will be able to observe how their documents are being used and this will subtly communicate information to the author about the actual use versus the intended use of their documentation. Writers can then compare the expected usage of their work versus its reality; and base maintenance efforts, where appropriate, on aligning the two views of the document's usage.

Hypothesis 4: When document maintenance is to be done on a software system, document Aura components can help prioritize the effort based on its usefulness. In particular, highly recommended artefacts for maintenance should be used most often (as measured by the LRFU / CRF metric), as well as should be poorly rated (which reflects low feedback from the artefact's audience).

Our justification for the above hypothesis is as follows: Documents that are highly referenced demonstrate some level of importance to the users, yet the consistently low feedback suggests that the user expected higher quality information. Unfortunately, the artefact was unable to provide such high quality information. Our perception of maintenance prioritization is not based on past maintenance activities (i.e. a high referential decay value) or the perceived importance (i.e. a high authority value). We have not included authority or referential decay because, as stated in Chapter 2, we believe documentation should communicate to its audience. Therefore, maintenance activities should focus on artefacts that are no longer able to effectively communication to its audience, but whose information continues to be needed by the software project team members.

6.3 Advantages of Using Document Aura

The primary advantage to using an approach such as document Aura to rank documentation is that it does not rely solely on content of the document. Instead, document Aura also considers the history of the document, the current settings of the system as well as the user of the system. To accommodate such circumstances, a document's Aura is not only a function of the documentation, but also a function of its users.

Document Aura is a combination of an artefact's usefulness, referential decay, as well as its authority. The weight of each element on the resulting Aura is not a static relationship. Rather, it very much depends on the environment in which the information is requested. The interpretation of a document's Aura should reflect the likelihood that a document is relevant at a particular moment in time under particular conditions. Drawing on existing knowledge in decision theory including Therrien in [50] and Michie et al. in [38] as well as the direct application to Engineering (such as Couvreur and Bresler in [16]) future work could investigate the appropriate weighting models and decision rules based on the hypotheses discussed in this chapter.

Document Aura attempts to combine environmental factors that affect the relevance of available artefacts and uses that information to better rank / recommend artefacts to interested users.

CHAPTER 7: A TOOL TO IMPROVE MAINTENANCE

This chapter discusses the details of Document Aura Calculator (DAC); a library that implements the document Aura algorithms discussed in Chapter 6 as well as providing a prototype interface for user experiments. The aim of this tool is to explore the architecture of a system designed to better integrate search, browsing and maintenance of documentation with actual software construction.

Our hope in building this tool is to validate the underlying concepts of document Aura and to more tightly couple documentation exploration with software exploration. The high level requirements of document Aura are outlined in detail in Chapter 5, Section 5.4; the models from which our implementation is based are available in Chapter 6.

7.1 A Plug-in for Eclipse

One of the fundamental requirements of our system is that it can integrate easily into the process of maintaining software; which apart from reading the documentation is comprised mostly of searching and writing code [47]. To accommodate this requirement we decided to integrate DAC into the Eclipse Project [19], an existing platform application that currently supports a feature-rich development environment for Java and C++.

Eclipse originated as a product by OTI and IBM and was first used by teams in those organizations building Integrated Development Environment (IDE) products. In essence, we created a plug-in to Eclipse to extend the platform features to include the Document Aura Calculator (DAC). This plug-in is packaged in a directory and installed by downloading our tool to the plug-in directory.

7.1.1 Integrating with Eclipse

The Eclipse Project is an open development that provides a framework in which tool integration is seamless and language neutral. One primary feature of Eclipse is its comprehensive developing

environment currently available for Java and C/C++. Eclipse is an appropriate platform to integrate tools directed towards software professionals such as DAC.

The Eclipse Standard Development Kit (SDK) as well as the integration of three *New Tool* plug-ins is illustrated below in Figure 7-1

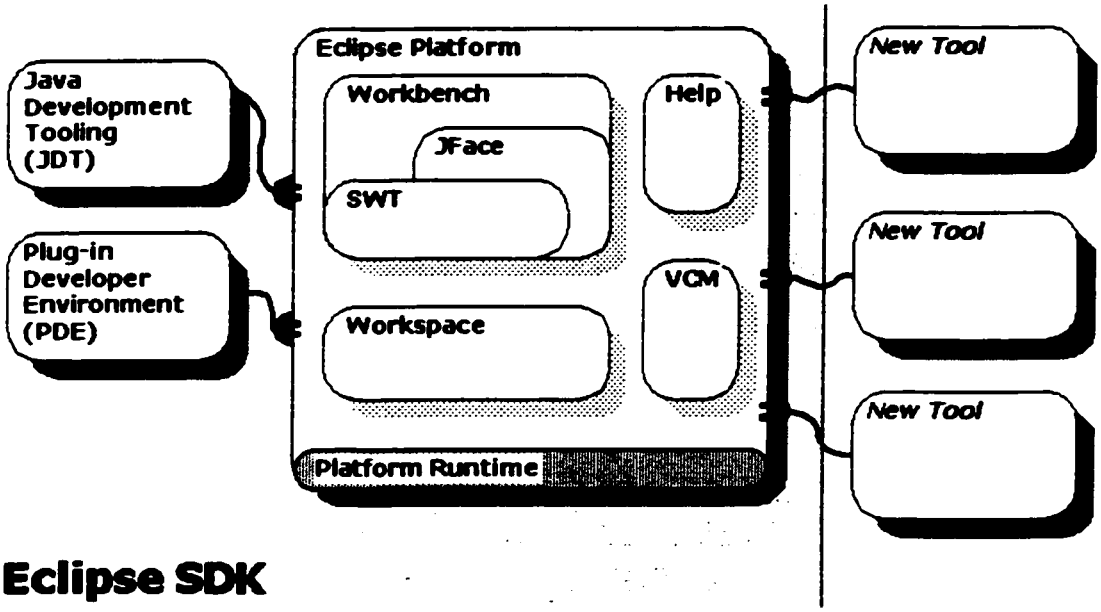

Figure 7-1: The Eclipse project architecture, available at [19]

Our plug-in integrates into Eclipse by providing an Eclipse user with additional DAC views which can be used to search, browse and maintain software documentation. In Figure 7-1, our document Aura plug-in would look similar to one of the *New Tool* blocks connecting the Eclipse Platform.

7.1.2 Screenshots of the Document Aura Eclipse Plug-in

The following section provides a sample screenshots of two DAC views integrated into Eclipse's Java Development Tooling (JDT) perspective.

Figure 7-2 shows the Result Set view integrated within Eclipse.

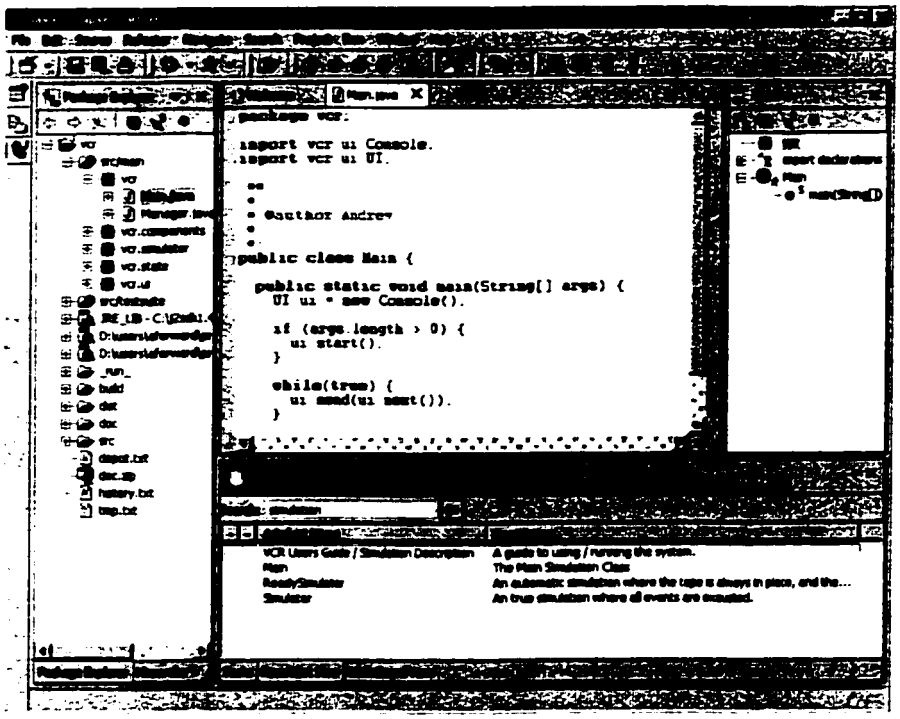

Figure 7-2: Document Aura's Result Set (bottom frame) integrated with Eclipse.

Figure 7-3 provides a closer view of the Result Set view shown above.

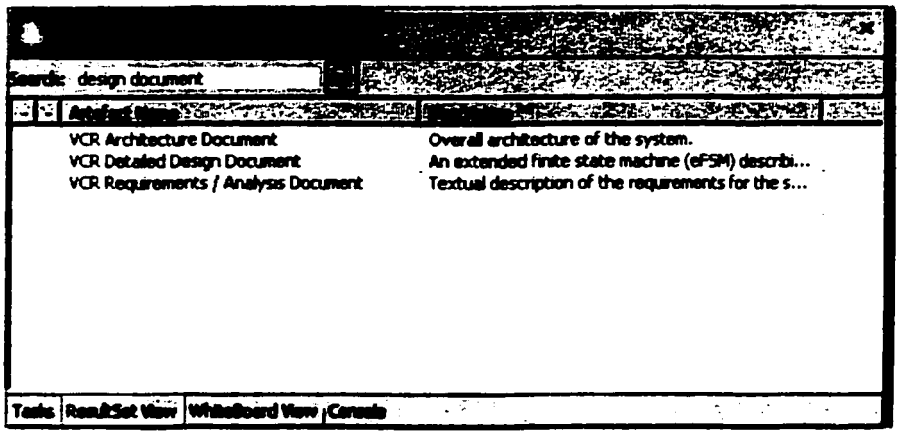

Figure 7-3: A Closer view of DAC's Result Set

The Result Set incorporates a simple text search where the results are ranked according to the document's CRF value (calculated using the LRFU algorithm discussed in Chapter 6, Section 6.1.1).

Figure 7-4 shows the White Board view integrated within Eclipse.

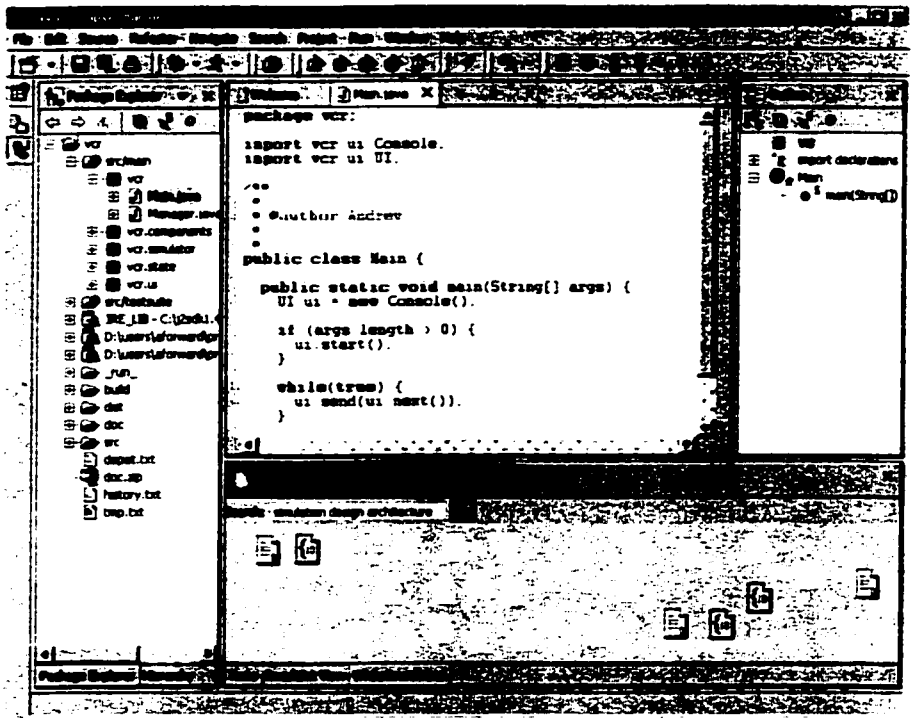

Figure 7-4: Document Aura's White Board (bottom frame) integrated with Eclipse.

The White Board incorporates the same simple text search feature as the Result Set. As opposed to ranking results, the White Board highlights potential results, as well as allows a user to manually group related documentation for potential future use.

Figure 7-5 provides a closer view of the Result Set view shown above.

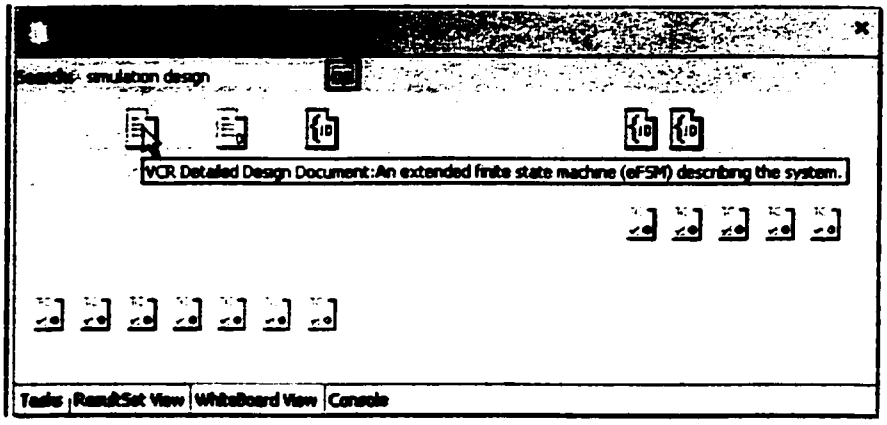

Figure 7-5: A Close view of Document Aura's White Board

DAC is also available as a standalone Java application. The interface combines the same Result Set and White Board views as seen in the Eclipse plug-in version. The standalone version is also based on Eclipse's Standard Widget Toolkit (SWT).

7.1.3 Implementation Details

Document Aura has been implemented at two distinct, yet important levels.

The first level is the API to manage the use and usefulness of a collection of artefacts. This includes the appropriate data and storage structures for the artefacts themselves, their access / maintenance history as well as the relationship amongst the artefacts. The API level also includes the individual Aura metrics calculators including usefulness, authority and referential decay as well as a simple text search (which is used with the authority metric).

The second level is the user interface integration with Eclipse. The user interface prototype primarily takes advantage of the Aura metrics calculators. In particular, the user can search the document repository and the result-set is generated based on the authority of the document (which combines the simple text search feature) and then ranks those documents based on the usefulness of the result-set.

7.2 Analyzing DAC's Design Using UML / XML

This section discusses the implementation details of Document Aura Calculator (DAC). Our analysis will present several UML diagrams (based on [35] and [42]) to provide both abstract and detailed views of the system. Our analysis begins by describing the main features of the system available to the user. Next we describe the architecture of the system using package and class diagrams. Following that, we illustrate one activity diagram to demonstrate the flow of messages involved in achieving one of the use case paths. Finally, we discuss how document Aura stores the usage and artefact data it collects from its users. The last section on DAC data is described using XML notation.

7.2.4 Interacting with Document Aura

Figure 7-6 highlights the primary use cases of the document Aura system. The requirements for these use cases were defined in Chapter 5, Section 5.4.

The three main ways an actor interacts with the DAC system is by searching for documentation and viewing the results, opening documentation to browse its content, as well as providing feedback about a particular artefact in the documentation.

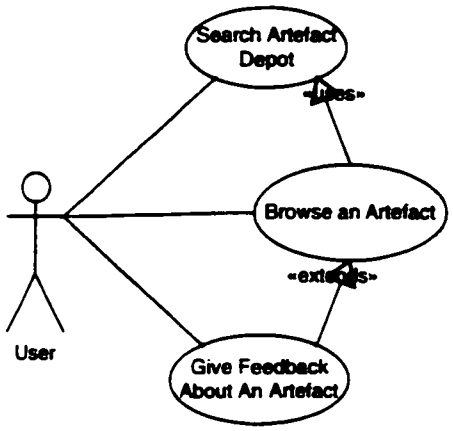

Figure 7-6: Document Aura's primary use cases

In general, a user may search the artefact depot for appropriate documentation. As well, the user may wish to browse an artefact which may follow from searching for it. Finally, a user may provide feedback about a document after having used it.

Although the system provides only three basic operations, the quality of system is based on the quality of the search results; which in turn is dependent on the system's tracking of document use and feedback regarding that use.

7.2.5 A Bird's Eye View of DAC's structure

Document Aura is defined by five main packages, each of which has a distinct yet important role. Figure 7-7 below outlines DAC's main packages.

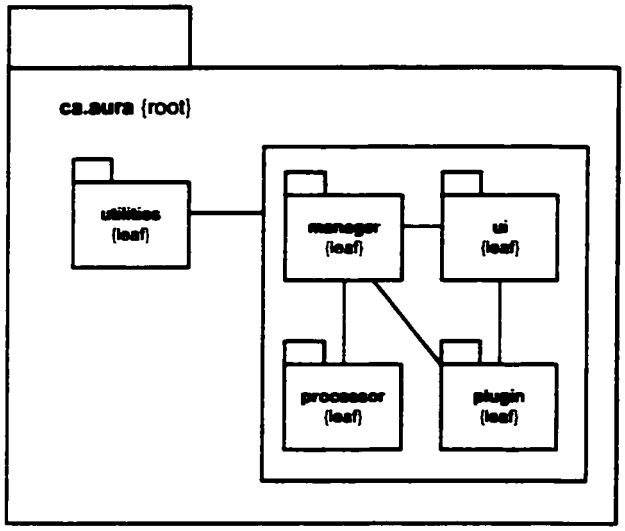

Figure 7-7: Document Aura package structure

The root package, **ca.aura,** has several leaf packages which are described below.

utilities

This package defines components which are available to and used by most other packages. This package defines the data structures commonly used to pass messages between objects. It also provides resources for storing and retrieving project properties and user data.

plugin

This package provides the necessary connection between the user interface components and the Eclipse Plug-in environment.

ui

This package implements an graphical interface in SWT, Eclipse's Standard Widget Toolkit. This package provides the necessary components to build a standalone application. As well, it can be combined to create an Eclipse Plug-in.

manager

The manager package coordinates and tracks the activities of the user. The package interacts heavily with the processor package.

processor

The package is responsible for processing the data made available by the manager. The processor

outputs are presented to the user. For instance, the document Aura algorithms are implemented in this package.

7.2.6 The Structure of Sequential and Parallel Algorithm Application

Ranking documentation based on its Aura is dependent on more than just the results of one document Aura algorithm (as described in Chapter 6). Instead, ranking is based on the environment in which user requests have been made. Based on our tool's implementation, the document Aura algorithms can be used in sequence or in parallel. Used in sequence, the outputs of first algorithm fuels the inputs to second, and so on. Used in parallel, the outputs are generated using the same inputs and then the results are normalized and merged to produce yet another perspective of the environment.

The document Aura algorithm classes shown below in Figure 7-8 provide insight into the structure used to allow sequential and parallel ranking. The design is appropriate for our needs as the rank algorithms make no judgment about the meaning of their results; they merely generate numerical values based on the rules of the algorithm. Other components can then use the algorithms results and infer meaning from their values.

Figure 7-8 highlights the main classes involved in generating results based on a particular search of the document repository. From Figure 7-8, we see that the main class is a ProjectManager. This class is composed of a Depot that is a collection of project artefacts, a History that tracks the usage of those artefacts and the Relationship among the Depot's artefacts. We also see that the ProjectManager can calculate various document Aura metrics. The ProjectManager can return the results sequentially in a SortIterator, or in parallel using the AuraWeightedIterator.

We should also note the relationship between a ProjectManager's aggregates. We decoupled the description of an artefact, its history and its relationships for several reasons. First this separation avoids contention when loading a project. We can safely load all of a project's artefacts prior to resolving its history and its relationships. Secondly, techniques that automate the indexing of the content and those that automatically relate artefacts to one another can be integrated separately into our work. More information about integrating other techniques to refine our work is discussed in Section 7.2.7, as well as in Chapter 9, Section 9.3.

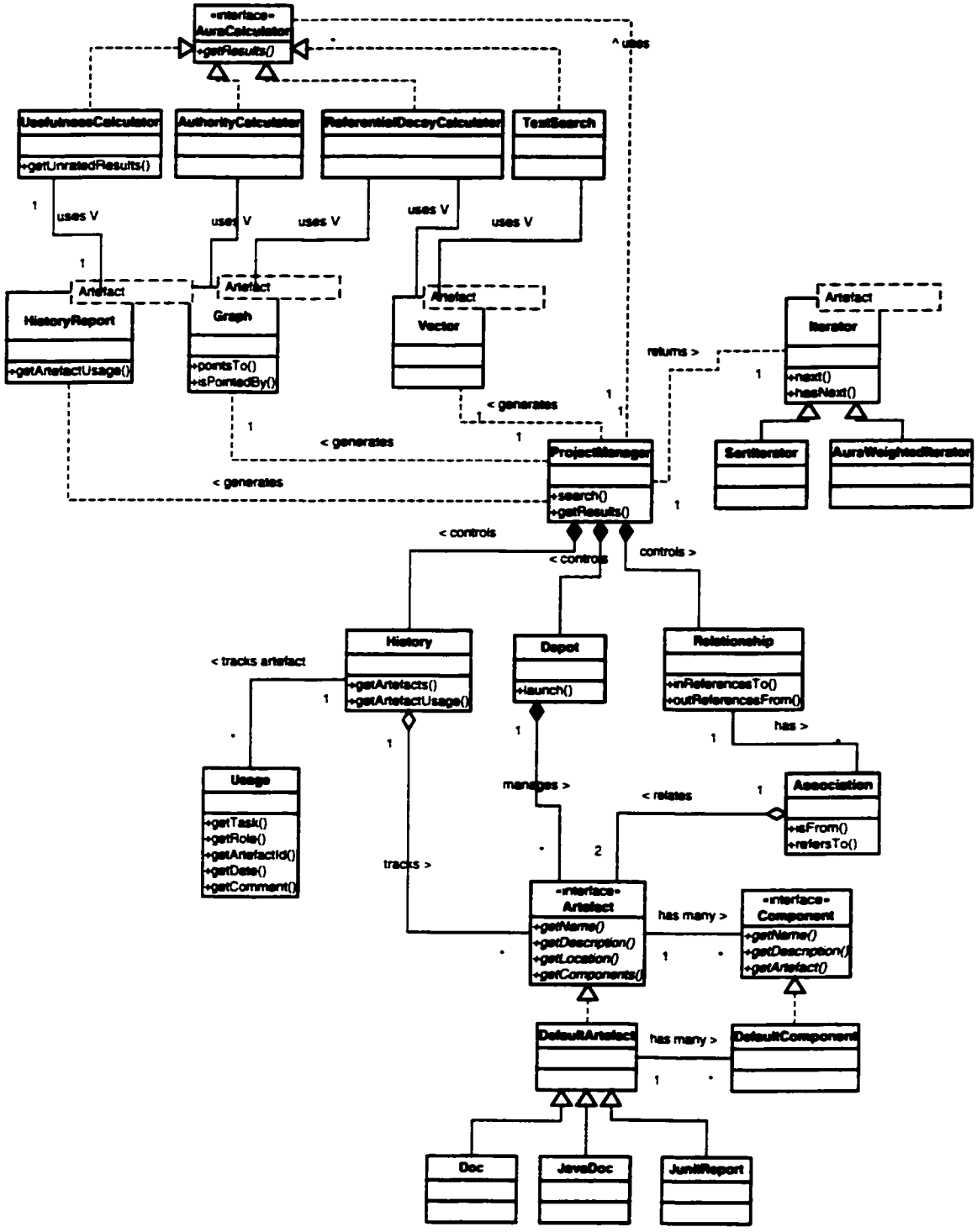

Figure 7-8: Document Aura class diagram

An example of sequential computation is demonstrated by the TextSearch and Authority algorithms. The TextSearch generates an output of appropriate artefacts based on a simple text search using the artefact's title and description. The ArtefactList generated by the ProjectManager is used by the TextSearch to a root set of artefacts to be used by Authority algorithm to further refine and rank the results.

Figure 7-9 illustrates one possible example of parallel computation.

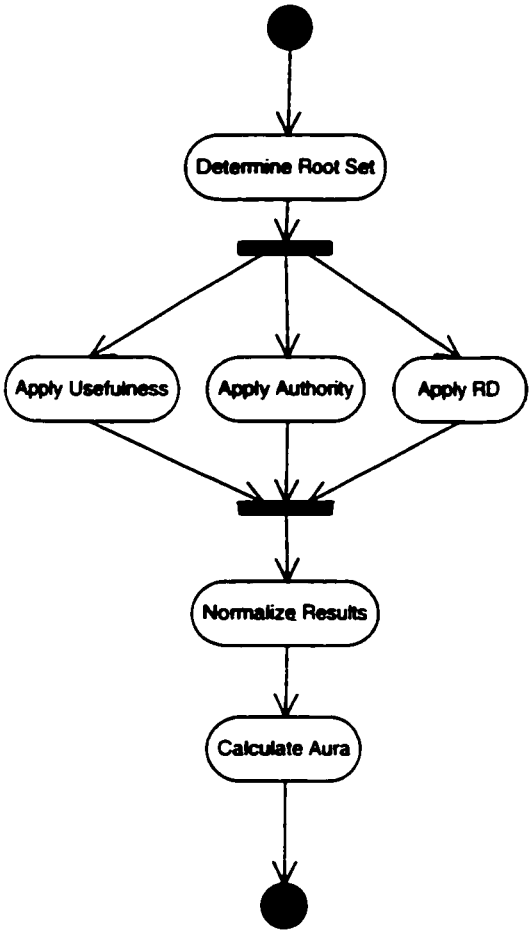

Figure 7-9: One possible parallel computation using DAC.

The example above is a combination of the LRFU, authority, and referential decay algorithms used to rank documentation. The process of normalizing the results involves applying a normalization scheme to allow an unbiased comparison and manipulation (such as using a percentile scale) of the sets of data (i.e. Usefulness, Authority and Referential Decay). The final Aura calculation would then be a suitable combination of the data above depending on the

particular circumstances (such as those outlined in Chapter 6, Section 6.2). For example, we may wish to have our results based 80% on high Authority, 15% on high Usefulness and only 5% on low Referential Decay. Once this activity is complete, the results can be sorted and displayed to the user.

7.2.7 Abstracting Artefact Knowledge In XML

To allow our tool to be sufficiently evaluated, we abstracted much of the knowledge about the software system and its documentation using a simple XML data layer. As discussed further in our future work (refer to Chapter 9, Section 9.3), in analyzing XML mark-ups of the real data, it becomes trivial for complimentary tools to contribute to DAC by contributing to its data.

As our work is only beginning to uncover the relevance of a technique such as document Aura, we feel it is premature to formalize our data needs in a DTD or XML Scheme. As future work in this area evolves, it will become more important to offer a formalized data model. For our purposes, an implicit understanding is sufficient and below in Figure 7-10, we have provided examples of the data structures used by DAC.

```
<depot>
 <artefact type="document" id="0" name="Architecture Document">
  <location>C:\architecture.doc</location>
  <description category="architecture">
    Overall architecture of the system.
  </description>
  <aura name="UML Class Diagram">
    <description category="uml" >The UML Class Diagram</description>
  </aura>
 </artefact>
  ...
</depot>
```

Figure 7-10: Data structure of an artefact depot (in XML)

As illustrated in Figure 7-10, an artefact has an attribute describing its type, such as a Word document, a webpage, a pdf, etc, as well as an attribute for the its name. Artefacts contain sub-elements including: a location, represented logically as a file (or more generally a URI) or physically such as a conference room white board; a description, which includes a brief introduction to the document as well as a description of its category such as requirements, design, architecture, etc; and finally Aura, which is a description of a particular section in a document (such as diagram, or figure).

Below in Figure 7-11 is the structure of a project's history. The history is a collection of usages of the available artefacts. The history tracks the task, its date of access as well as comments and / or a feedback rating (this rating is based on the 4-point scale introduced in Chapter 0).

```
<history>
 <usage artefact="0" task="create" date="2002.09.10"/>
 <usage artefact="1" task="browse" date="2002.09.12" >
 Learning about component Z.
   <rating>4</rating>
 </usage>
 <usage artefact="2" task="search" role="developer" date="2002.09.14">
  Searching unsuccessfully for item X, which I thought belonged in Y.
  <rating>1</rating>
 </usage>
 <usage artefact="3" task="maintain" role="developer" date="2002.10.20" />
 <usage artefact="3" task="validate" role="manager" date="2002.10.21" />
 ...
</history>
```

Figure 7-11: Data structure of a depot history (in XML)

Finally, our project maintains a collection of all relationships among artefacts (and their Aura components). Figure 7-12 provides an example relationship.

```
<relationship>
  <association>
    <from artefact="1" aura="UML Package Diagram" />
    <to artefact="2" />
  </association>
  ...
</relationship>
```

Figure 7-12: Data structure of a project's artefact relationships (in XML)

A relationship consists of several associations to and from artefacts (and optionally their Aura). For example, in Figure 7-12 there exists an association from an artefact's Aura (UML Package Diagram) to an entire other artefact.

We will discuss in further detail the advantages of abstracting the information above in XML in our future work section of Chapter 9, (see Section 9.3).

CHAPTER 8: VALIDATING DOCUMENT AURA

Chapter 6 introduced the documentation models that were later implemented in Chapter 7. This Chapter will expand on our approach to justifying the use of our models and validating our approach to their implementation.

8.1 Our Approach to Validation

Earlier in this thesis, we presented our initial study into the relevance of software documentation, as well as our investigation of existing means to manage and rank knowledge. We used this information to justify our approach to documentation maintenance.

As the final step in the work reported in this thesis, we conducted a second, simpler, empirical study to evaluate our tool and our document model. Lethbridge points out that, "the researcher would obtain far more accurate and quantitative information by recording and analyzing the developer's work practices" [37]. Our approach was to observe how software professionals interacted with our system as they attempted to complete several maintenance tasks on an existing system. Following the session, each participant responded to a questionnaire gauging several factors to help us qualitatively assess the usefulness of our research, and in particular the concepts behind document Aura.

We believe that our models to predict document relevance are applicable to most software projects, regardless of the processes and practices in place. However, asserting such a claim is difficult due to variables and parameters required to model software projects (and in particular interactions with software documentation). One approach would be to model software documentation parameters and then simulate the software environment [8][53]. We could then analyse the acceptability of our work by systematically altering the simulation parameters to reflect an assortment of software projects. Although our original work on software documentation (refer to Chapter 3, 4, and 5) does provide some insight into appropriate simulation parameters, a realistic simulation would amount to a thesis in itself. In particular, to develop a simulation is challenging, and acceptable results are valid only when appropriate models are used. Per Bak's [6] spherical cow is an example where a problem regarding the milk

production of farmer's herd of cows is resolved; provided we are dealing with a spherical cow (please note that cows are not spherical). Due to the risk involved, we have decided against validating our work using a simulation.

Instead, our approach is to involve real users on an actual software system. It would have been ideal to have data from a sufficiently large number of users so as to be able to achieve statistically significant results; however given that our thesis has already involved several different studies, we decided to limit ourselves to a smaller study that should reveal positive insights into the use and usefulness of our work and justify further, more quantifiable studies to be done by others. In other words, this experiment is not seeking definitive answers, but rather a qualitative reassurance of its applicability to the software engineering community. As well, we sought constructive feedback from the participants regarding our initial DAC implementation.

8.2 The Experiment Components

Our experiment involved four interrelated components:

First, we built a prototype user interface that exposes some of the elements of the Document Aura Calculator (DAC). For more information on DAC please refer to Chapter 7.

Second, we selected a software system that was used by a fourth year Software Engineering (SEG) course offered in January 2002 at University of Ottawa. The system was a VCR simulation tool composed of four main packages with over 1600 lines of code (without considering comments or test code). The system documentation included a requirements document, an architecture document, a detailed design document, several low level design documents (JavaDoc pages) and several quality assurance documents (JUnit reports).

Third, we integrated DAC and the VCR simulation into Eclipse. The VCR simulation software was added as an Eclipse project, whereas DAC was imported as an Eclipse plug-in.

Finally, we selected three students from the SEG course described above (all of whom are now working in industry) and two masters students unfamiliar with the VCR simulation. The participants' software industry experience ranged from 1 to 5 years and their software education experience was between 4 and 10 years.

Individual responses and identifying information have been withheld to protect confidentiality. The University of Ottawa's Human Subjects Research Ethics Committee approved the conducting of the study.

8.3 Research Method

Our session began with a brief re-introduction to the VCR simulator software, the development environment, as well as objectives of the research. In summary, we discussed the following:

- We do not expect you to consult the documentation merely because it is available; instead should you feel that the available documentation might help you complete your task, we ask that you use the available document Aura views.

- Our objective is to observe your usage pattern of the available documentation, as well as to give you a glimpse of how DAC can be tightly integrated to your development efforts.

The participants were asked to complete several maintenance tasks on the VCR simulator. The tasks were assigned at random and each maintenance session lasted around 30 – 45 minutes. During these sessions, we tracked which documents the users accessed to complete their task. Please refer to Appendix H for more details about the potential tasks that the users had to complete.

Following the maintenance session, the users answered a short questionnaire available in Appendix I. The questions were geared towards uncovering how the participant perceived the quality of the documentation, how they would allocate resources to improve the documentation, and to what extent the premise of DAC would influence or improve software and document maintenance.

8.4 Experiment Results

The following sections provide a qualitative analysis of the experiment data. Although not statistically significant, our preliminary investigation into the merits of using document Aura is positive.

8.4.1 Beginner Versus Advanced User Expectations

Following the experiment, the participants were asked to rank the quality of the documentation based on their own notion of quality and then based solely on it being complete and up-to-date. In all cases, for all documents except two, the users rated their own perception of quality as the same or higher than it being complete and up-to-date. This observation aligns with our original hypothesis that other factors beyond being complete and updatedness influence the quality of documentation.

Figure 8-1 compares the average quality rating for each individual based on personal perceptions of quality versus the quality of document based solely on being complete and up-to-date. Please note that only one participant's average quality based on personal perceptions was lower than that based on being complete and up-to-date.

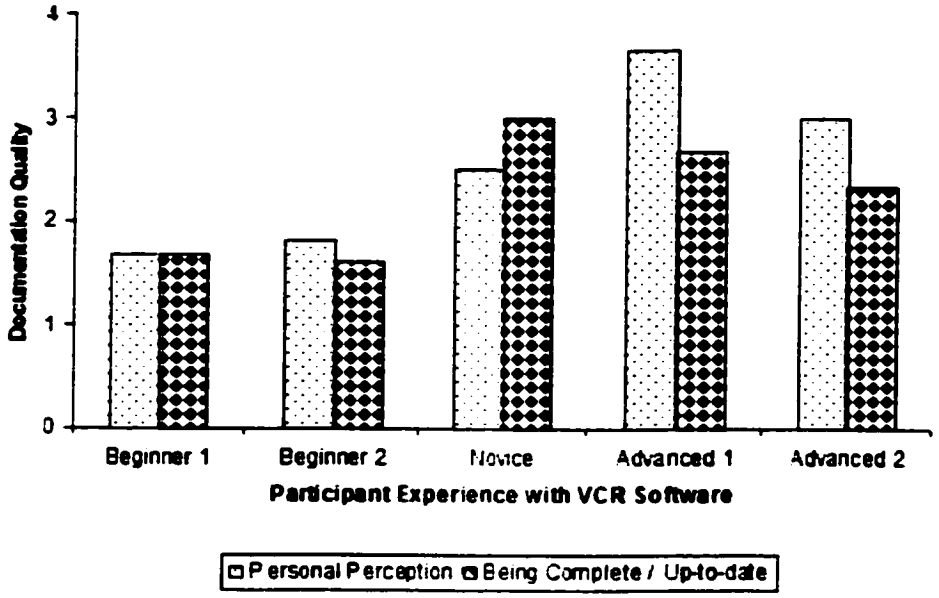

Figure 8-1: Personal Perception of document quality versus the quality of being complete and up-to-date (a higher value indicates higher quality)

It was also observed that the experienced users of the software rated the quality of the documents as good or excellent (except for the source code comments which were rated as poor by one of the

participants), whereas the inexperienced users rated most as poor, and at best some documents were good.

Figure 8-2 shows the perceived quality of beginner, novice and advanced users of the VCR simulation software.

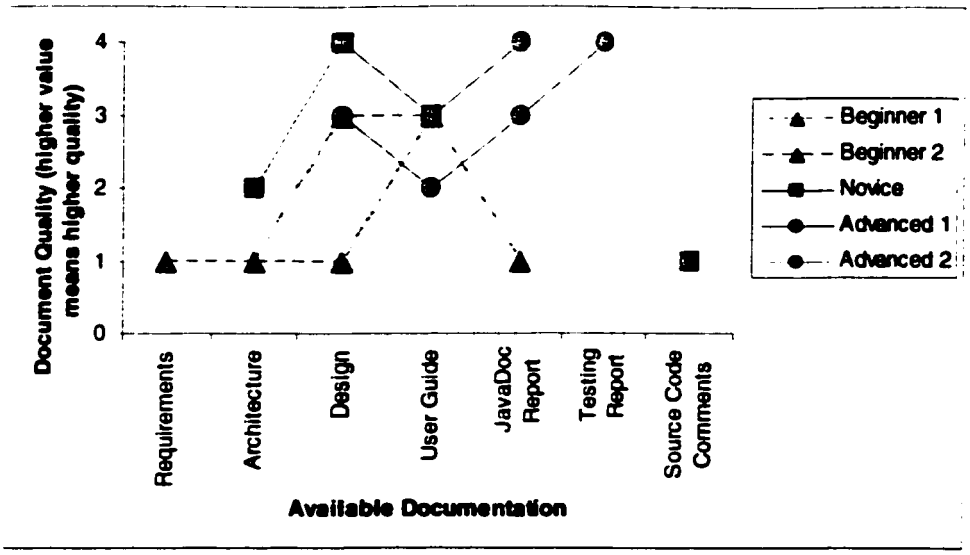

Figure 8-2: VCR simulation document quality (a higher value means higher quality)

An analysis of Cockburn [15] as described in Chapter 2 helps provide insight into these results. Cockburn describes three levels an individual experiences when learning something new. The beginner requires accurate information and recipe style instructions to follow. The beginner is not in a position to evaluate the information, but instead relies on it to accomplish his or her task. Meanwhile, the intermediate or expert user is better able to evaluate available alternative and can rely on experience when dealing with inadequate or insufficient information.

The hypotheses described in Chapter 6 that distinguish document Aura weighting based on learning the system and maintaining the system is supported by the experiment data above. In particular, we see that beginners are more likely to find the documentation to be poor quality and therefore relevance artefacts to have low referential decay.

8.4.2 Applying DAC to Prioritize Document Maintenance

The experiment results that our approach to document maintenance prioritization is appropriate are less suggestive than for beginner vs. expert user above. As the participants were accomplishing the software maintenance tasks, we tracked their use of documentation. The most-consulted documentation was JavaDoc, which is based on the source code and API of the software system.

Table 8-1 outlines the number of times a particular document was accessed by the participants, the average personal quality rating as well as the overall maintenance effort allocated for each document.

Table 8-1: Accessing a document, its quality and allocating effort to maintain it

	# Times Document Accessed	Average Quality	Maintenance Effort
Source Code / JavaDoc	15	2	40
Design	9	3	21
User Guide	9	2.8	0
Architecture	5	1.33	19
Requirements	2	1	8
Testing Report	1	4	4
New Doc: Software Features	0	0	5

To recall from Chapter 6, Section 6.2.3, we hypothesized that documents deserving maintenance were highly referenced, yet had low feedback ratings. Our results do indicate that the most likely candidate for update based on the participants' responses corresponds to our document Aura prediction. However, the subsequent documents do not convincingly follow the same pattern. An explanation for the high number of accesses to the User Guide is that even the advanced participants had not had experience with the system for several months and required some refreshing about the use of the system.

Nonetheless, the results do indicate that our approach would be suitable to define which documents should receive the least maintenance. Many participants did not consult the requirements document, and also did not assign any resources to maintain this document. Our experiment involved individuals whose task was one of maintenance, and, as one participant

noted "These [architecture and design documents] are the most important to understand the code that needs to be modified." Since the requirements document is not highly referenced, it requires less maintenance because it does not offer much insight to developers.

Many participants rated a particular document as poor, yet did not give that document a high priority for maintenance. This is most likely because the document was seldom used by the participant. For example, one participant rated the Requirements document as poor.

8.4.3 Promising Results Promoting DAC

Based on the premise of DAC, all users believed in the usefulness of our approach in gauging the relevance of a document as well as our prototype implementation that integrates the documentation process into the development process.

All participants at least somewhat agreed that the potential for DAC could be well adopted in the software development community. One participant's closing statement was "If I was developing in Java and I had to choose between two different IDEs, I would choose the one that included document Aura." In fact, all but one participant stated that indeed they would use a product such as DAC.

All participants felt strongly that an approach such as DAC would improve not only the use of documentation, but also its overall usefulness.

The initial response to DAC is very promising and as described more in Chapter 9 Section 9.3, future work invested in DAC could yield improvements in documentation maintenance as well as software maintenance in general.

8.5 Summary

Our experiment involved a target group of users that would consult documentation; namely software maintainers. As observed in the previous section, much of our work appears to have industrial merit. It is important to note that for such a system to be practical in a realistic software environment, the following key components of document Aura must be established.

First, the system should distinguish between the different roles of users interacting with the system. As seen in the results of our software survey, the use of software documentation varies based on software roles. For example, we found that the set of documents consulted by a project leader is somewhat different relative to a software developer. Therefore when ranking documentation, the system should also take into consideration the role of individuals using the system.

Second, the system should distinguish between the current task of a user. A similar result observed in our survey data, as well as in our experiment, was that the user's task influences his or her information needs. For instance, the browsing needs of a user were found to require accurate information, even more so then when searching the system.

Third, the system must integrate user feedback. A straight forward technique of weighing an individual's use of a particular document will improve the results of many document Aura ranking algorithms.

CHAPTER 9: SUMMARY AND FUTURE WORK

This chapter summarizes our research and our findings, as well as offers new direction for future work based on our ideas, implementations and discoveries.

Our initial hypothesis was that factors beyond merely being complete and up-to-date must be taken into consideration when predicting the relevance of a document. Our survey of software professionals provided the necessary justification to pursue other measures of relevance. In turn, we proposed the notion of an artefact's usefulness, referential decay and authority (*Aura*). Our later experiments with the Document Aura Calculator (DAC) supported our initial hypothesis.

9.1 Coping with Documentation

The choice between having either up-to-date and complete documentation that serves all your information needs, or else anything less than that seems obvious. However, if in order to make this decision we have to sacrifice other factors including time and budget constraints, or software construction and testing effort, the choice is less obvious. Our work has revealed that at an individual level, software professionals are willing and accepting of out-dated documentation.

Based on the above revelation, we investigated document attributes beyond completeness and correctness to find other ways to quantifiably predict the relevance of documentation. We therefore introduced the notion that we call document Aura; a dynamic refection of documentation quality based on several document attributes including usefulness, referential decay and authority. Using document Aura as a guide, our research suggests it will improve a document's use, its overall usefulness, as well as our ability to sufficiently maintain documentation.

9.2 Altering Software Moods

Based on observations from our survey of software professionals, we observed that the information needs of team members differ based on the member's task, as well as his or her role

in the group. For example, the information needs of a manager versus a developer are not the same. As well, the needs of a developer doing maintenance versus one that is learning the system also are different. The fact that information needs differ based on the task and user's role in the project is somewhat intuitive in nature, but is also supported by our data. We also used the above fact to justify our use of document Aura as a means to predict the relevance of a software document.

The primary advantage of using an approach such as document Aura to rank documentation is that it does not rely solely on the content of the document. Document Aura is a combination of an artefact's usefulness, referential decay, and authority. Aura also considers the environment under which information is being requested. Environmental factors that influence document Aura include elements such as the user's role in the software project, as well as his or her current task (such as searching, browsing or maintaining documentation).

Our work has introduced three document Aura weighting hypothesises based on our research into the use and usefulness of documentation to software professionals. In particular, we identified possible relationships when searching, browsing and maintaining documentation.

9.3 Future Work and Research Opportunities

Our work has introduced several new areas of research worth investigating. By exploring aspects of our work in more detail, one can validate our results and hypothesises, as well as build on our models and extend their application in the software community. The following sub-sections describe possible avenues of research resulting from our work.

9.3.1 Investigating the Documentation Process in Greater Detail

Based on our survey of software professionals, as well as the additional questions raised from our analysis of the data, we have compiled the following list of possible avenues for continued research.

- How do people use documentation in practice? More in-field research is required to substantiate some of our observations.

- How can documentation maintenance be improved? It would be useful to understand techniques and tools that improve this process.
- What effect does age have on a document's relevance? Our work introduced one measure of aging documentation based on referential decay, but how else does change affect relevance. For example, as a document ages, its relevance most likely decreases. Why? To what extent? How do external factors affect this decay in relevance to a software project?
- How else can we document a system and using what kind of artefacts? How effective are these methods at creating, maintaining, using and verifying such artefacts? Research in this area could widen our definition of documentation beyond just documents.

9.3.2 Extending and Refining Document Aura

Our use of document Aura has focused on document attributes beyond its content. Aura represents an array of document attributes that predict the relevance of a particular document based on its history, the current state of the system as well as the current demands of the user. Document Aura is not confined to the definition of usefulness, referential decay and authority. Future work looking to refine current techniques as well as extending the process with additional ones would benefit the entire process.

9.3.3 Document Aura in Practice

As noted in Chapter 8, our work with DAC was justified based on previous work in related fields, our survey of software professionals and our experiment with the DAC views. Our initial results validate the importance of work in the software industry. Future research could pursue additional analysis of our models and refine our approach based on further studies of the interactions with software professionals and documentation.

9.3.4 Combing DAC with Data Mining Techniques.

DAC uses a hypertext abstraction of the documentation to avoid the complications of documentation formats, styles and technologies. In abstracting the content of a document, we were able to model any artefact as documentation. Our approach avoids being tightly coupled to

the content of documentation. This decoupling facilitates the integration of independent techniques that specialize in analysing a document's structure and data.

For example, existing data mining techniques such as automated link generation (refer to work from Wilkinson and Smeaton [55] and Antoniol et al [5]) and information retrieval (an overview of document mining techniques is available at by Dixon [18]) can be applied to extend our work. Using an XML data layer to store all project data, complementary techniques can work together on the same data model with minimal concern about how each other works.

In application terms, the document Aura techniques would benefit from technologies in the following areas

- Content Summary. A data mining technique to index and summarize the contents of the available documentation.

- Link Generation. Another technique could relate artefacts to one another, including the source code.

- Improved text-based search techniques. Improving the root set used to calculate certain qualities of document Aura would improve the resulting document ranking.

Applying more informative and automatic techniques to summarize, relate and search a documentation base will improve the effectiveness the ranking of document Aura and integrating past research and practicalities in this area would improve the overall approach greatly.

9.4 Achieving Our Objectives

Our first objective, to uncover current practices of software documentation, confirmed that documentation was rarely maintained, somewhat consulted and almost always outdated. Our investigation also discovered that factors beyond merely being complete and up-to-date are taken into consideration in determining document quality.

Our second objective, to exploit the information above, uncovered the notion of document Aura. This metric is actually a combination of three metrics to predict the usefulness, referential decay and authority of a document. These factors are based on both present documentation conditions

as well as the document's history. Finally, document Aura requires calculating the influence of each factor above. The influence of the above metrics is dependent on the current needs and role of the user.

Our third object, to implement a tool based on the above, resulted in the creation of the Document Aura Calculator (DAC). Our tool was integrated into the Eclipse Project, a platform that offers a feature-rich development environment for Java and C++. DAC is to be used in conjunction with software construction.

Our final objective, to validate our models, was realized during a qualitative experiment involving DAC, a small software system and five software professionals.

Our work has analyzed the role of documentation from the perspective that software documents will never be entirely complete or up-to-date. Our work considers documentation in the realm of engineering where decisions are made in the presence of several factors, often conflicting, including time and money. We consider the metaphor that software engineering is a game with two objectives; first to deliver software and second to prepare for future delivery of software. Our models attempt to facilitate the use of incomplete and inconsistent information in an efficient manner. One must accept the limitations of software documentation as an intermediate work product whose role it is to allow software professionals to achieve their primary goals. Our work promotes both the primary goals of software engineering by integrating documentation with development and helping prioritize maintenance based on a document's Aura.

GLOSSARY

Agility "in business is dynamic, context-specific, aggressively change-embracing, and growth-oriented. It is not about improving efficiency, cutting costs, or battening down the business hatches to ride out fearsome competitive 'storms'. It is about succeeding and about winning: about succeeding in emerging competitive arenas, and about winning profits, market share, and customers in the very centre of the competitive storms many companies now fear" [26].

Automated Documentation Tools are software tools that transform / extract / summarize information about a software artefact automatically to create new artefacts of value; examples include JavaDoc and jUnitReport.

Artefact is an element of a project that reflects information about that project. This object may be concrete (hard-copy) or in digital format (soft-copy). It may also have a finite lifetime or be permanently archived. Examples include source code, detailed design documents, project specifications, etc.

Artefact Usefulness of a software artefact is a measure of how well a document's content can provide knowledge, information and / or insight to its reader with respect to other available artefact. Although a somewhat subject measure, there are several factors that can be tracked and measured to determine the usefulness of a document. Our measure of usefulness is a combination of an artefact's usage (how often is it references and by whom) and individual feedback from readers regarding the value of the artefact.

Artefact Value is an ordinal scale $V \in \{1,2,3,4\}$ that states to what degree an artefact offers helpful information and / or guidance. The values above correspond as follows; poor quality is 1, inadequate quality is 2, good quality is 3 and excellent quality is 4.

Broken window in software documentation is any artefact of a software system that elicits poor design, incorrect decisions or poor code [26].

Browsing is searching an artefact (or several artefacts) with the intent to learn overall concepts and not necessarily any one specific bit of information.

Documentation is an artefact whose purpose is to communicate information about the software system to which it belongs. Common examples of such documentation include requirement, specification, detailed design, and architectural documents, as well as low level design information such as comments in the source code. These documents are geared to individuals involved in the production of that software. Such individuals include managers, project leaders, developers and customers. We will not be considering end-user documentation.

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

Document Aura is a prediction of an artefact's relevance in a particular situation based on an artefact's usefulness, referential decay and authority (hence, Aura). A measure of Aura is a prediction based on the environment in which an artefact exists as well the conditions under which the prediction is requested.

Document Maintenance is the process of refactoring a document's content to improve its relevance. Maintenance activities include correcting errors, updating propagated changes and overall improving the document's content.

Document Management is the administrative process of organizing a document repository. This activity does not involve maintenance of a particular document's content. Instead, it involves the abstract coordination of documents into a useful and reliable whole; much like maintaining a useful and reliable filing cabinet. Tasks include appropriate naming conventions (for both individual documents and the folders that contain them), document filing (documents are in the right location), as well as archiving (or removing) older content.

DRY (Don't repeat yourself) A principle stating that every piece of knowledge must have a single unambiguous, authoritative representation within a system [26].

Good Enough in software production does not imply low or poor quality. Rather, it is a engineered solution that attempts to find a balanced and more realistic view of a project's external forces. Such forces include stakeholders, future maintainers, technical and limitations, budget

and timing constraints, managerial and corporate guidelines as well as personal state of mind. A more suitable term is Appropriate Software.

Study of Knowledge Management is the study of the characteristics or behaviour of workloads to a system. In particular, it is the study of access patterns to knowledge within a knowledge base and is based on recognition of access patterns through acquisition and analysis of past behaviour and history.

Law of Demeter for Functions attempts to minimize couple between software modules. Its purpose is to prevent a module from reaching into an object to gain access to a third object's methods. The law states that any method of an object should call only methods belonging to it, any parameters that were passed to the method, and any objects it created and any directly held component objects.

Law of Demeter for Documentation is an adaptation from the functions version above. The purpose is to prevent an artefact from citing information from another artefact that originates in a third artefact.

Least Frequently Used is a buffer management policy that ranks blocks by the frequency with which they are accessed. For example, when a buffer is full and a block must be removed, the block with the fewest references will be chosen.

Least Recently Used (LRU) is a buffer management policy that ranks blocks by the recency with which they have been accessed. For example, the least recently referenced block will be removed when the buffer is full. An improvement to this algorithm

Least Recently / Frequently Used (LRFU) is a buffer management policy that subsumes both the Least Recently Used (LRU) and Least Frequently Used (LFU) policies.

Model is a subset of documentation with a specific purpose, such as modelling the relationship among entities in a system. A model's lifespan typically lasts until the purpose is fulfilled (or the addressed problem is resolved); after which it can usually be discarded. The reason models are disposable is because the knowledge contained in them can be transferred to more permanent artefacts (such as the source code).

Orthogonality in a software system signifies the independence between or decoupling of two or more software artefacts. Orthogonality between two artefacts is best illustrated when changes to one artefact have no impact of the other artefact.

Parsing pattern the techniques that an individual uses to processes his or her surroundings. For example, the parsing pattern that usually defines the enjoyment of steak is texture, and not taste [15].

Perspective, in Eclipse, "defines the initial contents of the window action bars (menu and toolbar) and the initial set of views and their layout within a workbench page." [19]

Pragmatic Programmer derived from Latin to mean 'skilled in business', which itself is described from Greek meaning 'to do'. Hunt and Thomas [26], describe a pragmatic programmer as one who is agile (in the sense of being early to adopt and fast to adapt to new technologies and techniques; not necessarily an agile in the extreme programming sense), inquisitive, critical thinker, realistic and well-rounded.

Refactoring is a process of change for improvement's sake. Source code refactoring include changing variable names to more descriptive ones, improving algorithms to improve the use of system resources, un-coupling objects, etc. Documentation refactoring include adding more or better examples, updating content based on external changes, changing the organization of the document, etc.

Referential decay is a measure that reflects the likelihood of an artefact containing inconsistencies with other related artefacts, as well as the likelihood of being used in inconsistent ways in related artefacts.

Related Artefact is an artefact that references (either directly or indirectly) to another artefact, such as one artefact referencing another, or an artefact building upon the information in another artefact.

Relevance of a Software Document is a measure for how well suited a document should be to an individual. This measure takes into accounts both the appropriateness of a document as well as its usefulness. Relevance of a software document also accounts for the individual accessing the information, his or her purpose with the document and his or her role within the software project

Seeking is searching an artefact (or several artefacts) with the intent to learn / reference some particular information.

Specification is the process of taking a requirement and reducing it down to the point where a programmer's skill can take over. It is an act of communication, explaining and clarifying the world to remove (or at least reduce) major ambiguities.

Software Construction is the act of producing, maintaining and distributing software (usually involves a programming language, compiler),

Software Design includes all notations, representations and architectures of a software system (usually at a more abstract level of detail compared to software construction artefacts).

View, in Eclipse, "is a visual component within a workbench page. It is typically used to navigate a hierarchy of information (like the workspace), open an editor, or display properties for the active editor." [19]

REFERENCES

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

[2] Amento, B., L. Terveen, L., and W. Hill, *Does authority mean quality? Predicting expert quality ratings of web documents*. ACM SIGIR, p47, ACM Press, New York, NY. 2000.

[3] Ames, Andrea L. *Communicating Effectively with Interaction*. p 1 – 6, SIGDOC 2001, Sante Fe, New Mexico, USA, ACM Press, 2001.

[4] Angerstien, Paula. *Better quality through better indexing*, p 57, SIGDOC '85, Cornell University, Ithaca, New York, USA, ACM Press, 1985.

[5] Antoniol G. , G. Canfora, and A. De Lucia. *Recovering code to documentation links in OO systems*. In Proceedings of the Working Conference on Reverse Engineering, pages 136 – 144, 1999.

[6] Bak, Per. *How Nature Works, The Science of Self-Organised Criticality*. Copernicus Press, New York NY, USA, 1996.

[7] Bauer, Brian and David Parnas. *Applying Mathematical Software Documentation: An Experience Report*, p.273, Proceedings of the 10th Annual Conference on Computer Assurance (COMPASS'95), IEEE Computer Society Press, June 1995.

[8] Balci, Osman. *Guidelines for successful simulation studies (tutorial session)*, p.25-32, 1990 winter simulation conference proceedings on Winter simulation conference, New Orleans, Louisiana, United States, 1990.

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

[10] Botafogo, R., Rivlin, E., and Shneiderman, B. 1992. *Structural analysis of hypertext: Identifying hierarchies and useful metrics*, p 142–180ACM Trans. Inf. Sys. 10, 2 (Apr.), 1992.

[11] Briand, L, K. El Emam, S. Morasca. "On the Application of Measurement Theory to Software Engineering", Empirical Software Engineering- An International Journal 1996

[12] Brin, S. and L. Page. *The anatomy of a large-scale hypertextual web search engine*. Proc. of 7th WWW Conference, 1998.

[13] Chapin, Ned. *Software Maintenance Types – A Fresh View*, p247, International Conference on Software Maintenance (ICSM'00), 11-14 October 2000, San Jose, California, USA, Proceedings. IEEE Computer Society, 2000.

[14] Chapin, Ned. *Trends in Preserving and Enhancing the Value of Software*, p6, International Conference on Software Maintenance (ICSM'00), 11-14 October 2000, San Jose, California, USA, Proceedings. IEEE Computer Society, 2000.

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

[16] Couvreur, C. & Y. Bresler, "A Statistical Framework for Automatic Classification of Environmental Noise Sources," Submitted to Int. J. of Noise Control Eng., 1996.

[17] Curtis, Bill, Herb Krasner, and Neil Iscoe. *A Field Study of the Software Design Process for Large Systems*. Communications of the ACM 31(11):1268-1287, November, 1988.

[18] Dixon, Mark. *An Overview of Document Mining Technology*, available at http://citeseer.nj.nec.com/dixon97overview.html, 1997.

[19] The Eclipse Project, information available at http://www.eclipse.org/.

[20] Forward, A. Thesis Appendices and Additional Resources available at www.site.uottawa.ca/~aforward/thesis/ or www.site.uottawa.ca/~tcl/gradtheses/aforward/ .

[21] Forward, A. Survey data website available at [20]

[22] Forward, A. *Qualities of Relevant Software Documentation: An Industrial Study*, submitted to ICSE 2002, available at [20].

[23] Saul I. Gass, Karla L. Hoffman, Richard H. F. Jackson, Lambert S. Joel, Patsy B. Saunders: *Documentation for a Model: A Hierarchical Approach*. CACM 24(11), 1981.

[24] Goldman S, Nagel R, and Preiss K, *Agile Competitors and Virtual Organizations*, Van Nostrand Reinhold, 1995.

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

[26] Hunt, Andrew and David Thomas. *The Pragmatic Programmer*, Addison Wesley Longman, Inc. 2000.

[27] Janicki, Ryszard, David L. Parnas, and Jeffery Zucker. *Tabular representations in relational documents*. In C. Brink, editor, Relational Methods in Computer Science. Springer-Verlag, 1996.

[28] Jazzar, Abdulaziz and Walt Scacchi. *Understanding the requirements for information system documentation: an empirical investigation*, p268 – 279, COOCS `95, Sheraton Silicon Valley, California, USA, ACM Press, 1995.

[29] Jónsson, Björn T., Michael J. Franklin and Divesh Srivastava. *Interaction of query evaluation and buffer management for information retrieval*, P 118 - 129, ACM SIGACT, ACM Press, New York, NY, USA, 1998.

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

[31] Kalsbeek, Bill. *What is a Survey?*, ASA Series Section on Survey Research Methods, 1995.

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

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

[34] Lee, Donghee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. *On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies.* In SIGMETRICS99 [SIG99], pages 134 – 143.

[35] Lethbridge, Timothy C. and Robert Laganière. *Object-Oriented Software Engineering: Practical Software Development using UML and Java*, McGraw Hill, 2001

[36] Lethbridge, Timothy C., and Doug Skuce. *Beyond hypertext: knowledge management for technical documentation*, SICDOC, ACM Press, New York, NY, USA, 1992.

[37] Lethbridge, T., Sim, S. and Singer, J. (1998 July), *Studying Software Engineers: Data Collection Methods for Software Field Studies*, re-submitted May 2000 to Empirical Software Engineering available at www.site.uottawa.ca/~tcl/papers/.

[38] Michie D, D. J. Spiegelhalter & C. C. Taylor, eds, "Machine Learning, Neural and Statistical Classification", Ellis Horwood, 1994.

[39] Morasca, Sandro. Software Measurement, Università de l'Insubria, Italy, 1999.

[40] O'Neil, E. J. , P. E. O'Neil, and G. Weikum, *The LRU- K Page Replacement Algorithm For Database Disk Buffering*. ACM SIGMOD (pages 297-306), ACM Press, 1993.

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

[42] Pooley, Rob, Perdita Stevents. *Using UML, Software Engineering with Objects and Components*, Addison-Wesley, 1999.

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

[44] Sametinger, Johannes. *Reuse documentation and documentation reuse*. In Richard Mitchell, Jean-Marc Nerson, and Bertrand Meyer, editors, TOOLS 19: Technology of Object-Oriented Languages and Systems, pages 17--28. Prentice Hall, Paris, France, 1996.

[45] Scheff, Benson H. and Tom Georgon. *Letting software engineers do software engineering or freeing software engineers from the shackles of documentation.* p81 – 91, SIGDOC '88, Ann Arbor, Michigan, USA, ACM Press, 1988.

[46] Sim, Susan, Charles Clarke, Ric Holt, and Anthony Cox. *Browsing and searching software architectures.* In International Conference on Software Maintenance, Oxford, England, September 1999.

[47] Singer, J., Lethbridge, T., Vinson, N. and Anquetil, N. (1997), *An Examination of Software Engineering Work Practices*, pp. 209-223, CASCON '97, Toronto, October, 1997.

[48] Singer, J., Lethbridge, T. and Vinson, N. (1998), *Work Practices as an Alternative Method to Assist Tool Design in Software Engineering*, University of Ottawa, Computer Science Technical Report TR-97-08, 1998.

[49] Slinn, Michael. *Cost-Benefit Analysis.* http://tomcat.mslinn.com/JspExplorer/info/documentationCost.html

[50] Therrien, C. "Decision, Estimation and Classification, An Introduction to Pattern Recognition and Other Related Topics", John Wiley & Sons, 1989.

[51] Thomas, Bill, Dennis Smith and Scott Tilley. *Documentation for software engineers: what is needed to aid system understanding?*, p 235 – 236, SIGDOC '01, Sante Fe, New Mexico, USA, 2001.

[52] Tilley, Scott. *Documenting-in-the-large vs. documenting-in-the-small.* In Proceedings of the 1993 IBM/NRC CAS Conference (CASCON '93), (Toronto, Ontario; October 25-28, 1993), pages 1083--1090, October 1993.

[53] Tuncer I. Ören, *Concepts and criteria to assess acceptability of simulation studies: a frame of reference*, Communications of the ACM, v.24 n.4, p.180-189, April 1981.

[54] Walker, J. H. *Document Examiner: Delivery interface for hypertext documents*, Hypertext '87 Proceedings, p 307—323, 1987.

[55] Wilkinson R. and A. Smeaton. *Automatic Link Generation.* ACM Computing Surveys, 31(4es), Article No. 27, December 1999.

APPENDICES

Appendix A **The introduction to the on-line documentation survey**

Refer to [20].

Appendix B **Informed consent for the on-line documentation survey**

Refer to [20].

Appendix C **Sample on-line question format for the documentation survey**

Refer to [20].

Appendix D Document Survey Questions

Please refer to Appendix C for the questions presentation.

Question 1 of 50 (Multiple Choice Multiple Answer Question)

Which types of software documentation do you write / edit / verify? Check all that apply. For this survey references to the term 'software documentation' imply documents geared for software developers that support a given system and NOT for end users.

Select all that apply from:

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

Question 2 of 50 (Rating Question)

In your experience, who has the prime RESPONSIBILITY to CREATE and MAINTAIN the following types of software documentation? Select 1: Customer(s) / Client(s). Select 2: Manager / Project Leader. Select 3: Software Architects. / Sr. Developers. Select 4: Jr. Developers. Select 5: Technical Writers.

Rate for each of the following:

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

Question 3 of 50 (Rating Question)

In your experience, who has the prime RESPONSIBILITY to VERIFY and VALIDATE the information in the following types of software documentation? Select 1: Customer(s) / Client(s). Select 2: Manager / Project Leader. Select 3: Software Architects. / Sr. Developers. Select 4: Jr. Developers. Select 5: Technical Writers.

Rate for each of the following:

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

Question 4 of 50 (Rating Question)

In your experience, when changes are made to a software system, how long does it take for the supporting documentation to be updated to reflect such changes? Rate 1: Updates are NEVER made. Rate 2: Updates are RARELY made. Rate 3: Updates are made within a few MONTH of the changes. Rate 4: Updates are made within a few WEEKS of the changes. Rate 5: Updates are made within a few DAYS of the changes.

Rate for each of the following:

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

Question 5 of 50 (Rating Question)

In your experience, how effective are the following types of software documents in reflecting the true state of a software system. Rate between one (1) as completely USELESS and five (5) as extremely EFFECTIVE.

Rate for each of the following:

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

Question 6 of 50 (Rating Question)

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

Rate for each of the following:

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

Question 7 of 50 (Rating Question)

In your experience, how effective do you find the available software documentation for a software project in the following circumstances. Rate between one (1) as completely USELESS and five (5) as extremely EFFECTIVE.

Rate for each of the following situations:

- When LEARNING a software system. When MAINTAINING a software system. When TESTING a software system.

- When other developers are UNAVAILABLE to answer my questions. When explaining / answering questions about the system to MANAGEMENT or CUSTOMERS.
- When looking for BIG-PICTURE information about the software system.
- When looking for IN-DEPTH information about the software system.
- When working with a NEW software system. When working with an ESTABLISHED / MATURE software systems.

Question 8 of 50 (Rating Question)

In your experience, how well maintained is supporting software documentation in the following type of software projects (please note, we are referring to both the quality and frequency of the updates). Rate between one (1) as NOT MAINTAINED at all and rate five (5) as extremely WELL MAINTAINED.

Rate for each of the following situations:

- New or recent projects. Mature projects with new functionality.
- Maintenance projects (where the software is being supported but relatively few new features are added).
- Agile or lightweight projects (for example eXtreme projects, extremeprogramming.org).
- Open source, public domain projects.

Question 9 of 50 (Rating Question)

In your experience, how important is each of the following items in helping to create effective software documentation (requirements, design, architecture, ...) to its audience. Rate the LEAST important item as one (1) and MOST important item as five (5). All other items are somewhere in between.

Rate for each of the following items:

- Length (not too short, not too long)
- Availability (ability to retrieve the most current version)
- Organization (table of contents, categorized, sub-categorized, etc)
- Navigation (internal / external links, actual links or just references)
- Document structure (arrangement of text, tables, figures and diagrams)
- Document's format (i.e. Microsoft Word, Note Pad, Visio, Html, Pdf)
- Author
- Content (the information that a document contains)

- Type (requirements, specification, detailed design, architectural document)
- Influence from management / project leaders / other developers to use it
- Spelling and grammar
- Writing Style (choice of words, sentence and paragraph structure)
- Extent to which it is up-to-date
- Use of modelling diagrams (UML, SDL, etc)
- Use of examples (how to extend or customize a feature).

Question 10 of 50 (Free-Form Question)

What types of software artefacts (ie. use case maps, sequence diagrams, CRC cards, finite state machines) do you find MOST effective for software documentation? Briefly justify how these artefacts are effective and explain how they are used.

Question 11 of 50 (Free-Form Question)

What types of software artefacts do you find LEAST effective for software documentation? Briefly justify your answer.

Question 12 of 50 (Rating Question)

How relevant are the following factors in causing software documentation to be out of sync with the system it describes. Rate between one (1) as a completely IRRELEVANT factor and five (5) as an extremely RELEVANT factor.

Rate for each of the following factors:

- Time constraints on developers
- Budget constraints on the project
- High costs of maintaining documentation is not worth the effort
- Rapid changes in requirements
- Rapid staff turnover
- Team members do not believe in documenting their code
- Team members are unmotivated to document their code
- Team members see little benefit in always maintaining supporting documents

Question 13 of 50 (Likert-Scale Question)

If software documentation could be graded (based on several factors such as appropriate length, readability, accuracy, ease of use, and the extent to which it is up to date) then it might be more useful to me.

Question 14 of 50 (Likert-Scale Question)

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

Question 15 of 50 (Likert-Scale Question)

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

Question 16 of 50 (Likert-Scale Question)

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

Question 17 of 50 (Likert-Scale Question)

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

Question 18 of 50 (Likert-Scale Question)

Tools to view and browse software documents are bulky and inefficient.

Question 19 of 50 (Likert-Scale Question)

Tools to view and browse software documents facilitate my work.

Question 20 of 50 (Likert-Scale Question)

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

Question 21 of 50 (Likert-Scale Question)

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

Question 22 of 50 (Likert-Scale Question)

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

Question 23 of 50 (Likert-Scale Question)

The cost of maintaining most software documents greatly outweighs the benefits of having such documents up to date.

Question 24 of 50 (Likert-Scale Question)

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

Question 25 of 50 (Likert-Scale Question)

Tools that help extract information from source code (for example JavaDoc) are extremely powerful for creating supporting documentation.

Question 26 of 50 (Likert-Scale Question)

Software documentation concentrates too heavily on high level issues rather than the important implementation details.

Question 27 of 50 (Likert-Scale Question)

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

Question 28 of 50 (Likert-Scale Question)

Much more effort is required to format / prepare software documentation than the effort of providing the actual content for the documents.

Question 29 of 50 (Likert-Scale Question)

I practice (or am trying to practice) agile software (agilemanifesto.org) techniques (i.e. eXtreme Programming (extremeprogramming.org) and Test First programming) and believe in the effectiveness of unit testing.

Question 30 of 50 (Likert-Scale Question)

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

Question 31 of 50 (Likert-Scale Question)

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

Question 32 of 50 (Likert-Scale Question)

I would rather refactor / update / debug / test a system than document it in its present (and flawed) state.

Question 33 of 50 (Likert-Scale Question)

We use a useful configuration management system to maintain our software documentation.

Question 34 of 50 (Likert-Scale Question)

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

Question 35 of 50 (Likert-Scale Question)

Only a few software documents are personally useful to me.

Question 36 of 50 (Free-Form Question)

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

Question 37 of 50 (Free-Form Question)

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

Question 38 of 50 (Rating Question)

Relative to past projects, please compare the amount of documentation being produced in your current project compared to the amount of software delivered (i.e.. lines of code, features developed, etc). Rate between one (1) as extremely LESS documentation and five (5) as extremely MORE documentation.

Rate for each of the following:

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

Question 39 of 50 (Rating Question)

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

Rate for each of the following criteria:

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

Question 40 of 50 (Rating Question)

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

Rate for each of the following criteria:

- Maintenance (Is is easier to keep the documents up to date?)
- Writing style (Is it easier to read and understand?)
- Navigation (Is it easier to move between documents?)
- Searching (Is it easier to find the information you need?)
- Updated (Are the documents more up to date?)

Question 41 of 50 (Multi-Choice Single Answer Question)

What is the size of your current (or recently completed) project in KLOCs.

Select one of the following:

- < 1 KLOC (KLOC = 1000 lines of code)
- between 1 and 5 KLOCs
- between 5 – 20 KLOCs
- between 20 – 50 KLOCs
- between 50 – 100 KLOCs
- over 100 KLOCs

- N/A

Question 42 of 50 (Multi-Choice Single Answer Question)

How long have you been working in the software field?

Select one of the following:

- < 1 year
- 1 - 4 years
- 5-10 years
- > 10 years

Question 43 of 50 (Free-Form Question)

What type of products / services does your company offer?

Question 44 of 50 (Multi-Choice Multiple Answer Question)

What is / are your current job function(s)?

Select all that apply:

Manager, Project Leader, Software Architects, Sr. Software Developer, Jr. Software Developers, Technical Writers, Software Support, Quality Assurance, Student, Other, None of the above

Question 45 of 50 (Multi-Choice Multiple Answer Question)

What are some past job functions you have held?

Select all that apply:

Manager, Project Leader, Software Architects, Sr. Software Developer, Jr. Software Developers, Technical Writers, Software Support, Quality Assurance, None of the above

Question 46 of 50 (Multi-Choice Multiple Answer Question)

What type of development process(es) does your company / manager recommend?

Select all that apply:

No defined process, Waterfall model, Incremental model, Iterative model, Agile process, Test first strategies, Rational unified process, Personal software process, Clean Room Approach, Code and Debug, Internal Process, Other, N/A

Question 47 of 50 (Free-Form Question)

Describe to what extent you are successful at following the recommended development process(es).

Question 48 of 50 (Rating Question)

To what extent does your software team do the following activities. This list is described by Joel Spolsky as The Joel Test. More information is available online at joelonsoftware.com. Rate between one (1) for NEVER and five (5) for ALWAYS.

Rate for each of the following:

- Use source control (ie. CVS system)
- Have daily builds
- Maintain a bug database
- Fix bugs before writing new code
- Keep an up-to-date schedule
- Have a specifications document for your project
- Have new candidates for the development team write code during the interview
- Have testers
- Conduct hallway usability tests (ie. Grab someone from the hallway to try the newly added feature)

Question 49 of 50 (Free-Form Question)

Do you use tools to automatically document your system? If so, which tools and to what extent.

Question 50 of 50 (Free-Form Question)

Briefly describe your companies policy for software documentation? To what extent and how successfully is it followed?

Appendix E Condensed Survey Results

Question	Option	Average	Std Dev	Mode	Median	Max	Min	Percentage	Count
1	a	--	--	--	--	--	--	67%	32
	b	--	--	--	--	--	--	73%	35
	c	--	--	--	--	--	--	60%	29
	d	--	--	--	--	--	--	52%	25
	e	--	--	--	--	--	--	67%	32
	f	--	--	--	--	--	--	54%	26
2	a	2.07	1.04	2	2	5	1	--	27
	b	2.72	0.89	3	3	5	1	--	25
	c	3.04	0.35	3	3	4	2	--	25
	d	3.50	0.67	4	3.5	5	2	--	22
	e	2.96	0.34	3	3	4	2	--	27
	f	3.15	1.04	2	3	5	2	--	20
3	a	1.70	0.95	1	1	5	1	--	27
	b	2.08	1.00	2	2	4	1	--	25
	c	3.08	0.70	3	3	4	2	--	25
	d	3.13	0.87	3	3	4	1	--	23
	e	2.93	0.68	3	3	4	1	--	27
	f	2.70	1.34	2	2	5	1	--	20
4	a	2.78	1.22	2	2	5	1	--	27
	b	3.12	1.13	2	3	5	2	--	25
	c	2.96	1.27	2	3	5	1	--	25
	d	2.96	1.36	2	2	5	1	--	23
	e	3.04	1.15	2	3	5	1	--	26
	f	3.74	1.25	5	4	5	1	--	23
5	a	3.11	1.26	3	3	5	1	--	28
	b	3.50	1.03	4	3.5	5	2	--	26
	c	3.19	1.13	3	3	5	1	--	26
	d	3.00	1.18	2	3	5	1	--	24
	e	3.54	1.04	4	4	5	1	--	28
	f	3.67	1.27	5	4	5	1	--	24
6	a	3.14	1.41	4	3.5	5	1	--	28
	b	3.85	1.29	5	4	5	1	--	26
	c	3.19	1.36	3	3	5	1	--	26
	d	2.96	1.31	3	3	5	1	--	25
	e	3.68	1.36	5	4	5	1	--	28
	f	3.28	1.40	3	3	5	1	--	25
7	a	3.64	1.13	4	4	5	1	--	28
	b	3.04	0.96	3	3	5	1	--	26
	c	3.46	1.17	4	4	5	1	--	26
	d	3.38	1.20	4	3.5	5	1	--	26
	e	2.96	1.19	3	3	5	1	--	27
	f	3.39	1.31	5	3	5	1	--	28
	g	2.93	1.21	2	3	5	1	--	28
	h	3.57	1.03	4	4	5	2	--	28
	i	3.00	1.05	3	3	5	1	--	28

Question	Option	Average	Std Dev	Mode	Median	Max	Min	Percentage	Count
8	a	3.39	1.03	3	3	5	1	--	28
	b	2.96	1.00	3	3	5	1	--	28
	c	2.39	1.10	2	2	5	1	--	28
	d	2.63	1.07	3	3	5	1	--	19
	e	3.29	1.38	2	4	5	1	--	21
9	a	3.15	1.17	4	3	5	1	--	27
	b	4.19	0.83	4	4	5	2	--	27
	c	3.85	0.91	3	4	5	2	--	27
	d	3.26	1.20	2	3	5	1	--	27
	e	3.26	0.94	3	3	5	2	--	27
	f	2.42	0.99	2	2	4	1	--	26
	g	2.63	1.31	1	3	5	1	--	27
	h	4.85	0.36	5	5	5	4	--	27
	i	3.78	1.05	4	4	5	1	--	27
	j	2.62	1.24	3	2.5	5	1	--	26
	k	2.93	0.83	3	3	4	1	--	27
	l	3.26	0.94	3	3	5	1	--	27
	m	4.35	0.69	5	4	5	3	--	26
	n	3.44	1.09	4	4	5	1	--	27
	o	4.19	0.79	4	4	5	2	--	27
10		--	--	--	--	--	--	--	36
11		--	--	--	--	--	--	--	31
12	a	4.07	1.15	5	4.5	5	1	--	28
	b	3.39	1.07	3	3	5	2	--	28
	c	3.11	1.37	3	3	5	1	--	28
	d	3.68	1.28	5	4	5	1	--	28
	e	3.07	1.36	2	3	5	1	--	27
	f	3.86	1.27	5	4	5	1	--	28
	g	4.04	1.35	5	5	5	1	--	28
	h	4.00	1.22	5	4	5	1	--	28
13		3.37	1.29	4	4	5	1	--	46
14		2.38	1.40	1	2	5	1	--	45
15		3.38	1.29	4	4	5	1	--	47
16		3.30	1.04	4	4	5	1	--	47
17		2.79	1.12	2	2	5	1	--	47
18		2.71	1.24	2	2	5	1	--	45
19		3.73	1.03	4	4	5	1	--	45
20		3.60	1.23	4	4	5	1	--	45
21		3.96	0.93	4	4	5	2	--	47
22		2.89	1.37	2	2	5	1	--	47
23		2.59	1.42	2	2	5	1	--	46
24		3.38	1.33	4	4	5	1	--	47
25		3.76	1.19	5	4	5	1	--	45
26		2.39	1.06	2	2	5	1	--	46
27		3.83	1.20	4	4	5	1	--	47
28		2.66	1.48	1	2	5	1	--	47
29		3.76	1.32	5	4	5	1	--	38

Question	Option	Average	Std Dev	Mode	Median	Max	Min	Percentage	Count
30		3.70	1.11	4	4	5	1	--	40
31		4.13	1.00	5	4	5	1	--	46
32		3.50	1.19	4	4	5	1	--	44
33		2.98	1.55	4	4	5	1	--	44
34		3.11	1.29	4	4	5	1	--	46
35		3.48	1.22	4	4	5	1	--	46
36		--	--	--	--	--	--	--	41
37		--	--	--	--	--	--	--	33
38	a	3.19	1.17	3	3	5	1	--	26
	b	3.08	1.10	3	3	5	1	--	24
	c	2.88	1.03	3	3	5	1	--	24
	d	2.79	1.02	3	3	5	1	--	24
	e	3.12	0.93	3	3	5	1	--	25
	f	3.33	1.05	3	3	5	1	--	24
39	a	3.30	1.06	4	3	5	1	--	23
	b	3.28	0.98	3	3	5	2	--	25
	c	3.16	0.90	3	3	5	1	--	25
	d	3.59	0.85	4	4	5	2	--	22
	e	3.39	1.12	3	3	5	2	--	23
	f	3.26	1.10	4	3	5	1	--	23
40	a	3.39	0.94	3	3	5	2	--	23
	b	3.42	0.93	4	4	5	1	--	24
	c	3.38	0.71	3	3	5	2	--	24
	d	3.38	0.88	3	3	5	2	--	24
	e	3.25	0.85	3	3	5	1	--	24
41	a	--	--	--	--	--	--	0%	0
	b	--	--	--	--	--	--	2%	1
	c	--	--	--	--	--	--	27%	13
	d	--	--	--	--	--	--	13%	6
	e	--	--	--	--	--	--	10%	5
	f	--	--	--	--	--	--	25%	12
	g	--	--	--	--	--	--	19%	9
42	a	--	--	--	--	--	--	0%	0
	b	--	--	--	--	--	--	23%	11
	c	--	--	--	--	--	--	29%	14
	d	--	--	--	--	--	--	46%	22
43		--	--	--	--	--	--	--	44

Question	Option	Average	Std Dev	Mode	Median	Max	Min	Percentage	Count
44	a	--	--	--	--	--	--	25%	12
	b	--	--	--	--	--	--	29%	14
	c	--	--	--	--	--	--	35%	17
	d	--	--	--	--	--	--	40%	19
	e	--	--	--	--	--	--	10%	5
	f	--	--	--	--	--	--	21%	10
	g	--	--	--	--	--	--	6%	3
	h	--	--	--	--	--	--	19%	9
	i	--	--	--	--	--	--	2%	1
	j	--	--	--	--	--	--	8%	4
	k	--	--	--	--	--	--	6%	3
45	a	--	--	--	--	--	--	29%	14
	b	--	--	--	--	--	--	46%	22
	c	--	--	--	--	--	--	21%	10
	d	--	--	--	--	--	--	48%	23
	e	--	--	--	--	--	--	65%	31
	f	--	--	--	--	--	--	25%	12
	g	--	--	--	--	--	--	33%	16
	h	--	--	--	--	--	--	27%	13
	i	--	--	--	--	--	--	4%	2
46	a	--	--	--	--	--	--	15%	7
	b	--	--	--	--	--	--	29%	14
	c	--	--	--	--	--	--	29%	14
	d	--	--	--	--	--	--	31%	15
	e	--	--	--	--	--	--	27%	13
	f	--	--	--	--	--	--	15%	7
	g	--	--	--	--	--	--	17%	8
	h	--	--	--	--	--	--	2%	1
	i	--	--	--	--	--	--	2%	1
	j	--	--	--	--	--	--	19%	9
	k	--	--	--	--	--	--	29%	14
	l	--	--	--	--	--	--	10%	5
	m	--	--	--	--	--	--	2%	1
47		--	--	--	--	--	--	--	34

48	a	4.29	1.33	5	5	5	1	--	28
	b	3.15	1.43	5	3	5	1	--	27
	c	3.86	1.18	5	4	5	1	--	28
	d	3.43	0.96	3	3	5	2	--	28
	e	3.26	0.98	3	3	5	1	--	27
	f	3.22	1.34	2	3	5	1	--	27
	g	2.08	1.32	1	2	5	1	--	24
	h	3.61	1.26	4	4	5	1	--	28
	I	2.30	1.07	1	2	4	1	--	27
49		--	--	--	--	--	--	--	34
50		--	--	--	--	--	--	--	28

Appendix F Detailed Free-Form Survey Results

Refer to [20].

Appendix G Detailed Survey Results

Refer to [20].

Appendix H Document Aura Experiment Tasks

TASK – Run the Simulation

What types of simulations can you run with the VCR software?
How would you run these programs?
Now, please run the VCR simulation in all available modes.

TASK – Extend UI Interface

How do you quit the simulation?
Add the ability to quit the VCR simulation by sending a "quit" command to the console.

TASK – Event Handling

How would the command "inject play timeout" be handled by the system?
Add functionality to the system to allow for multiple events such as that above to process.

TASK – Stop Pausing After 5 minutes

If the system is paused for more than 5 minutes, then the VCR should stop.

TASK – Reset Tape length

What is the default tape length?

Set the default tape length to 90 minutes.

Appendix I Document Aura Experiment Follow-Up Questionnaire

Please rate your comfort level with the following items.

		Beginner	Novice	Intermediate	Advanced	Expert
Java						
Eclipse IDE						
VCR Simulation Software						
JUnit Testing						
Software Patterns						

How many years of software education experience do you have? _____
How many years of in the software profession do you have? _____

You only have time to update two of the documents, which would you choose?

First Choice: ____
(1) Requirements
(2) Architecture
(3) Design
(4) User Guide
(5) Source Code Comments

Second Choice ____
(1) Requirements
(2) Architecture
(3) Design
(4) User Guide
(5) Source Code Comments

Why did you choose these two documents? Also, why did you choose not to update the remaining three documents?

Overall, what are your thoughts about the documentation available in this system? From each group, please select the most appropriate answer.

Amount of Documentation:

- ☐ Not enough documentation.
- ☐ An appropriate amount of documentation.
- ☐ Too much documentation.

Additional Comments

Integrity of the Documentation

- ☐ Documentation contained a lot discrepancies between the source code, causing some problems
- ☐ Same as above, except it didn't really cause too many problems.
- ☐ Same as above, except the discrepancies did cause a lot of problems
- ☐ Did not really link the documentation to the source code, or other documents
- ☐ I did not notice that many integrity mistakes

Additional Comments

Documentation Sufficiency

- ☐ Documentation was sufficient to my needs
- ☐ Documentation wasn't poor, just inappropriate, as it did not have the information I was looking for
- ☐ Documentation was poor and did not address my needs
- ☐ The source code was adequate enough, so I did not consult the documentation as much
- ☐ The documentation was ambiguous and I had to infer a lot about the expectations from the system.

Additional Comments

Based on your own perception of quality, how would you rate poor (1), inadequate (2), good (3), excellent (4) the documentation in this system? Put N/A for documents that you did not reference (or notice) the particular document.

- ☐ Requirements
- ☐ Architecture
- ☐ Design
- ☐ User Guide
- ☐ JavaDoc Report
- ☐ Testing Report
- ☐ Source Code Comments

Based on the correctness and completeness of the documentation, how would you rank the following documents?

- ☐ Requirements
- ☐ Architecture
- ☐ Design
- ☐ User Guide
- ☐ JavaDoc Report
- ☐ Testing Report
- ☐ Source Code Comments

For the following questions, please base on answers based on your experience with the software system, and that you have just been asked to change a particular functionality of the system. For instance, change behavior of the Rewind mechanism.

Suppose it will take you 20 units to accomplish this task, how much time would you spend on the items below to accomplish your task?

- [ ] Requirements
- [ ] Architecture
- [ ] Design
- [ ] User Guide
- [ ] JavaDoc Report
- [ ] Testing Report
- [ ] Source Code / Comments
- [ ] Test Code / Comments

Based on the content of the documentation, and it's ability to provide you with appropriate information to accomplish the task at hand, how would rank the follow documents? Rate between poor (1), inadequate (2), good (3), excellent (4).

- [ ] Requirements
- [ ] Architecture
- [ ] Design
- [ ] User Guide
- [ ] JavaDoc Report
- [ ] Testing Report
- [ ] Source Code / Comments
- [ ] Test Code / Comments

For the following questions, please base on answers based on your experience with the software system, and that you have just been asked to improve that available documentation.

Suppose it will take you 20 units to accomplish this task, how much time would you spend on the items below to accomplish your task?

- [ ] Requirements
- [ ] Architecture
- [ ] Design
- [ ] User Guide
- [ ] JavaDoc Report
- [ ] Testing Report
- [ ] Source Code Comments
- [ ] Test Code / Comments
- [ ] Create a new document (Specify Type _____)
- [ ] Create a new document (Specify Type _____)
- [ ] Create a new document (Specify Type _____)

The following questions are in regards to our tool, Documentation Aura. Based on the premise of Documentation Aura, please provide your opinion on the following statements. Please leave the question blank if you have no strong opinion.

	No	Not Really	Somewhat	Indeed
It is a feasible project from a technology perspective.				
It is a feasible project from a tool adoption perspective.				
It is a unique project that I would like to see developed further.				
It will help improve the use and usefulness of documentation while making changes to a software system.				
It will help prioritize documentation maintenance tasks.				
Once a product oriented interface is applied to Documentation Aura, I would consider using it.				

Additional Comments

Appendix J Document Aura Experiment Results

Refer to [20].