Writing and Reading Software Documentation: How the Development Process may Affect Understanding

Remco C. de Boer
Department of Computer Science
VU University Amsterdam
the Netherlands
remco@cs.vu.nl

Hans van Vliet
Department of Computer Science
VU University Amsterdam
the Netherlands
hans@cs.vu.nl

Abstract

The effectiveness of documentation within a development process is determined by the way in which the intentions of the authors correspond to the expectations of the potential readers. Ideally, the members of a development team share a certain understanding of (the role of) the different types of documentation. However, since one's expectations of a document are personal, and part of a tacitly formed mental model, we can expect different levels of shared understanding between different development team members.

We elicited and analyzed the mental models of software documentation from eight members of a single development team. We found indeed different levels of shared understanding between different people. To our surprise, the levels of shared understanding within the team appear closely tied to the development process employed. From Conway's law we know that an organization's structure is mirrored in the structure of the software that the organization produces. Our findings suggest that the organization's development process may likewise be mirrored in the extent to which a development team shares a common frame of reference. Hence, the development process followed may have implications for the effectiveness with which development knowledge can be shared through software documentation.

1 Introduction

During software development, important knowledge is shared between different members of the development team. Such knowledge can be shared for example face-to-face in more or less formal settings, but most software development projects rely to at least a certain extent on written documentation as a means of sharing knowledge. As a result, many projects end up with huge stacks of documents, in which one can easily lose track of the information needed. A question one may have is how effective this document-driven knowledge sharing is, and what its (in)effectiveness is based on.

The effectiveness of documentation within a development process is determined by the way in which the intentions of the authors correspond to the expectations of the potential readers. In a typical software development process, many different kinds of documents are produced and consumed at various points in time. The contents of those documents necessarily exhibit a certain amount of overlap. People may lose track of the meaning of individual documents; which information it contains and what its role is in the development process. When the expectations of the consumers of the documentation drift too far from the intentions of its producers, the ultimate consequence might be a need to rediscover already documented knowledge. In such a situation, customers for example may need to explain their situation and requirements over and over again to different parties in the development process.

If an author's work is part of a team effort, as is the case in software development, the intention an author has cannot be detached from his expectation of (the intention of) the work by other authors from that team. The author of, for example, a requirements specification may intend that document to be slightly more prescriptive than, say, a software architecture description, but only when the author has a certain expectation of the prescriptiveness of the latter. In this sense, an author's intent can only be expressed in terms relative to his expectation of the other documents.

Ideally, the members of a development team share a certain understanding of (the role of) the different types of documentation. Such a shared understanding would imply that an author's intentions are perfectly clear to others in the development team. However, since the way in which one interprets and tacitly models the world (of which software documentation is but a small part) is co-determined by such

CHASE'09, May 17, 2009, Vancouver, Canada
978-1-4244-3712-2/09/$25.00 © 2009 IEEE

intangible things as personal background, experience, and social interactions, we can expect different levels of shared understanding between different team members.

Suppose I am looking for a particular piece of knowledge in a set of documents. Chances are that I will not read all documents, especially when the number of documents is high. Instead, I will rely on a mental model of the documentation that I tacitly built up. This model tells me, for instance, whether some documents contain too much detail, or are instead too vague, to satisfy my knowledge need. This is far more likely to be a matter of heuristics than of exact science, and I may even apply it to documents that I've never even read (or, as a consequence, will ever read). My understanding of the documentation, represented by this model, may be entirely different from the understanding of others – including that of the documents' author(s). Consequently, I may fail to find important information, or conversely the author may fail to relay important information to me.

We elicited and analyzed the mental models of software documentation from eight members of a single development team. As expected, we found different levels of shared understanding between different people. To our surprise, however, the shared understanding within the team highly resembles the development process followed within the organization. It seems that people who work on similar tasks share much of the way in which they look at the project's documentation, but this shared understanding is lost when the number of 'process hops' increases.

From Conway's law we know that the structure of an organization is mirrored in the structure of the software that organization produces. Our findings suggest that the organization's development process may likewise be mirrored in the extent to which a common frame of reference exists within a development team. Hence, the development process followed may have implications for the effectiveness with which development knowledge can be shared through software documentation.

The remainder of this paper is organized as follows. In the next section, we briefly introduce personal construct theory – the psychological theory upon which our research is based – and some of its associated methods. In § 4 we describe the methodology we followed. In § 5 we introduce the development organization in which our experiment took place. § 6 presents the results of the experiment and reconstructs the shared understanding within the development team. In § 7 we conclude this paper with a discussion of the results we obtained.

2 Personal Construct Theory

The methodology we employ builds on personal construct theory, a constructivist theory of personality and psychology developed by George Kelly [7]. Personal construct theory says that every individual observes the world in terms of bipolar 'constructs', such as 'good-bad', 'right-wrong', et cetera. The use and definition of those constructs may differ from person to person. My comprehension of the construct 'good-bad' may not be the same as yours. Together, a person's constructs make up a personal mental model of (part of) the world.

Kelly proposed not only a psychological theory, but also a method to systematically elicit and analyze personal constructs. This method, called the Repertory Grid Technique (RGT), can be seen as "an attempt to stand in others' shoes, to see their world as they see it, to understand their situation, their concerns" [4].

The way RGT works is that it presents a subject with 'triads' of (i.e., groups of three) elements from the domain under investigation. In our case, since we're interested in the way people see software documentation, those elements are documents. When the three elements are presented, the subject is asked in which way two of them are alike and the third one is different. This identifies a bipolar construct or axis that is apparently part of the subject's mental model. Given, for example, the triad {use case specification, test charter, software architecture description} a development team member might indicate that the use case specification and test charter have something in common, and that the software architecture description is different. The next question then would be what it is that makes the two documents similar and the other different, to which the answer could be 'the use case specification and test charter both show low-level details, and the software architecture description provides a high-level overview'. This indicates that the construct 'low-level details/high-level overview' is part of this person's mental model.

Typical constructs that one may expect for software documentation include 'abstract-concrete', 'overview-detail', 'specification-implementation' and many more. Recall that those constructs are personal, hence their use and interpretation may differ between individuals. Two people who both distinguish 'low-level detail' documents from 'high-level overview' documents may disagree on the level of detail present in individual documents. Therefore, in a second step of RGT, the subject is asked to arrange all documents, including the three just used to elicit the construct, on a 5-point scale – from 'low-level detail' (1) to 'high-level overview' (5) – which provides insight into the way in which the subject applies the construct. This process is repeated until the method elicits no more new constructs.

The elements, constructs and rankings are kept in a tabular format called a 'rank order grid'. In this grid, rows represent personal constructs, columns represent documents, and the cells contain a number from 1 to 5 that corresponds to the rank of a particular document on a particular construct. The data from this grid can be further analyzed.

The 'Focus' algorithm, for example, is a straightforward algorithm for cluster analysis of RGT data, which is simple enough that it can even be used to calculate element and construct clusters by hand [5]. The algorithm involves a pairwise difference calculation and subsequent iterative reordering of constructs and elements "so that the differences between the resulting pairs are minimised", eventually producing a similarity matrix from which construct or element clusters can be derived.

Shaw and Gaines present another method to analyze repertory grid data [13]. The method entails a Focus-like comparison of constructs, but instead of comparing constructs within the same grid, this method compares constructs from different grids. Through this comparison, the relations between the 'conceptual systems' of two subjects can be analyzed. Shaw and Gaines define four types of relations: consensus, correspondence, conflict, and contrast.

When there is consensus, 'experts use terminology and concepts in the same way', i.e. two people use the same construct in a similar fashion. For example, two development team members may both use the construct 'abstract-concrete' to distinguish documents, If they do so such that by and large documents that are abstract according to one team member are also abstract according to the other, and documents that are concrete according to the one are also concrete according to the other, there would be consensus among the two regarding the construct 'abstract-concrete'.

Two people may also 'use different terminology for the same concepts'. This is called correspondence and happens, for example, when two team members both distinguish documents based on their level of abstraction, but one of the team members uses the construct 'abstract-concrete' for this distinction while the other uses 'high level-low level'.

Conflict takes place when 'experts use the same terminology for different concepts', e.g. when two people use the same construct 'abstract-concrete' but disagree on which documents are abstract and which ones are concrete. And, finally, contrast is when 'experts differ in terminology and concepts', i.e. when two constructs from their respective rank order grids have nothing in common whatsoever.

3 Related Work

Several researchers have successfully applied personal construct theory and RGT to the software development process. The main applications reported in the literature can be summarized as 'understanding the user' and 'understanding the developer'. Niu and Easterbrook [11], for example, report on the use of RGT in goal-oriented requirements engineering to compare and clarify stakeholders' terminology. And Karapanos and Martens [6] use RGT as a means to characterize user diversity in (software-intensive) product development. Examples of the use of RGT to understand developers include the work of Baddoo and Hall [1], who analyze the roles of practitioners in software process improvement, and Young et al. [10], who analyze personality characteristics of the members of an agile development team. In our work, we use RGT to analyze the extent to which development team members have a shared understanding of a software product's documentation.

The topic of shared understanding, or shared mental models, between development team members has received considerable attention. Klimoski and Mohammed argue that such a model "implies a variety of content", which means that "the content of shared mental models might reference representations of tasks, of situations, of response patterns *or* of working relationships" [8]. In other words, there may be various types of shared understanding between team members of which the shared understanding of software documentation is only one. As far as we know, our study is the first to relate shared understanding to production and consumption of software documentation. Still, our findings seem to be in line with those of other researchers such as Levesque et al. [9] who show – through a focus not on documentation but on team process and expertise – how an increase in role differentiation leads to less interaction and eventually to a decrease in shared understanding.

4 Methodology

Based on the personal construct theory and analysis techniques introduced in § 2, we constructed a method to analyze the mental models of various development team members regarding documentation. The method progresses through a number of steps: the first step is elicitation of the personal mental models of software documentation, in the next steps assessment of the levels of shared tacit and shared explicit understanding between team members takes place, and finally the results of those two assessments are combined to determine the aggregate level of shared understanding between team members.

The distinction between shared tacit and shared explicit understanding is taken from the field of knowledge management. In knowledge management, 'tacit knowledge' pertains to the kind of knowledge one possesses but cannot easily express. 'Explicit knowledge', on the other hand, is the knowledge that can be expressed in various forms, such as conversations, electronic communication, or writing (cf. [12]). Likewise, with 'shared tacit understanding' we mean the 'alignment' of two mental models without the individuals necessarily being able to express the exact foundation upon which this alignment is based. With 'shared explicit understanding' we denote the possibility for two individuals to talk about the way they see software documentation in terms that each of them can understand, i.e., relate to their own mental models. Fig. 1 graphically depicts these

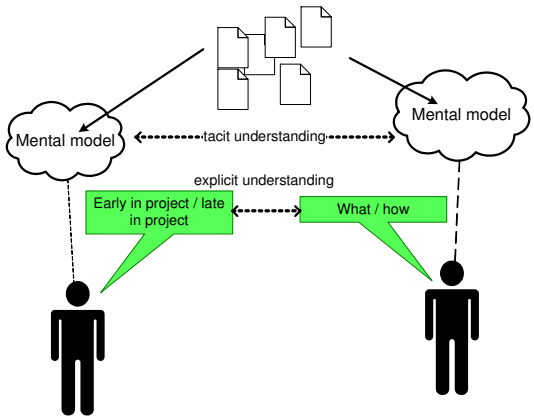

Figure 1. Shared tacit and explicit understanding between individuals

two types of understanding.

The steps of our method are further described below. This description includes the analysis methods we use, and their relation to personal construct theory from § 2.

Elicit personal mental model We first elicit, using the Repertory Grid Technique, the individual expectations and impressions regarding the documentation. This provides us for each of the development team members with personal constructs that are used to distinguish and classify documents. Application of the Focus algorithm leads to a clustering of documents, which captures a personal mental model of the software documentation that may differ from person to person.

Determine level of shared tacit understanding A shared tacit understanding indicates that different team members have similar mental models of the document set as a whole, although it says nothing about how the individual mental models have been built up. We therefore take the alignment of two personal mental models as a measure for the level of shared tacit understanding between two team members, disregarding the constructs upon which those mental models have been based.

To compare the alignment of different mental models of documentation, we determine to what extent the document clusters the team members perceive are correlated. This correlation can be calculated using the so-called Simple Mantel Test (cf. [2]). A high correlation found by this test indicates that two mental models lead to similar document clusters, in other words that two individuals share a tacit understanding of the documentation. A correlation ≥ 0.5 is generally considered to be large (cf. [3]), so we use a correlation of 0.5 as the lower bound to determine whether two team members have a shared tacit understanding of the documentation.

Determine level of shared explicit understanding In addition to the levels of shared tacit understanding, we also determine the levels of shared explicit understanding between the team members. With a shared explicit understanding we mean that two development team members distinguish between documents in a similar fashion. For this distinction they need not necessarily employ the same terminology. For example, one team member may distinguish between abstract and concrete documents in much the same way as another team member distinguishes between early and late documents. Even though they do not use the same terminology, the distinction can in principle be expressed and communicated, provided that the two team members recognize that the terminology they use is synonymous. Shared explicit understanding is complementary to shared tacit understanding; a shared explicit understanding says nothing per se about similarities between the tacit mental models.

To determine the level of shared explicit understanding, we determine to what extent there is 'consensus' and 'correspondence' between two team members, i.e., the extent to which constructs from two mental models are being used in the same way, with the terminology used being the same (consensus) or different (correspondence, cf. § 2). Shaw and Gaines propose a 80% similarity threshold as the lower bound for correspondence and consensus [13]. We consequently use this threshold to determine whether there is a shared explicit understanding between two team members.

Determine level of shared understanding By combining the results of steps 2 and 3, we determine the structure of shared (tacit and explicit) understanding in the team. Whenever two individuals meet the thresholds of both shared tacit and shared explicit understanding, we say that they have a 'shared understanding'. A shared understanding of the documentation implies similar expectations and/or intentions regarding individual documents (with respect to the other documents, cf. § 1).

5 Development team

We conducted our study within a development team that is part of a large software development organization. Software development is organized in what the organization calls 'development streets'. A development street provides a standardized environment (i.e., processes and tools) for developing a particular type of software. There are for example different development streets for the development of Java-based software and for the development of dotNet-based software. The project we observed took place in one of those development streets.

Fig. 2 shows the main stakeholders in the development street who produce or consume documentation. It also shows the documents with dependencies where information from one document is used in the production of another. The process can be summarized as follows. The business analyst starts off with determining the stakeholders,

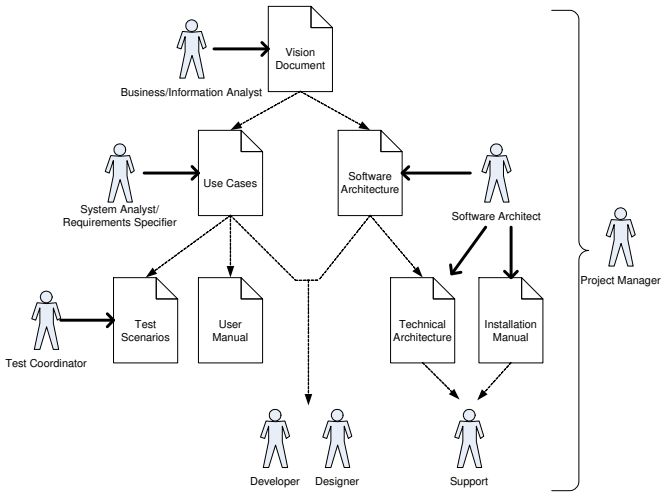

Figure 2. Development street

goals, and high-level functionality of the envisioned system, and reports the result of his analysis in a vision document. From the contents of this document requirements are derived, both functional and non-functional. The former are captured in use cases by the system analyst, while the latter are part of the software architecture description written by the software architect. Based on the use cases, the test coordinator devises test scenarios. The use cases are also reflected in the user manual (for which there usually is no dedicated author). From the software architecture description, the technical architecture is derived. This document, together with the installation manual, is primarily used by support personnel (for installation, configuration, etc.). Both technical architecture and installation manual are written by the software architect. Finally, the use cases and software architecture documents are used by designers and developers in the construction of the actual software.

Fig. 2 originated from early discussions with one of the company's senior employees. Based on this figure, we invited a total of 8 individuals who all participated, in different roles, in the same development project. All invitees participated in our study. In a later phase, we independently verified with the participants that the process depicted in Fig. 2 is indeed the process followed within this team[1].

At the time of our study, the team was working on the second release series (2.x) of the software. Together with the project manager, we selected a representative subset of software product documentation to be used in the experiment. We had to make this selection since the number of documents produced in the project far outnumbered the maximum number of elements (± 20) that can feasibly be assessed by the repertory grid technique.

[1] We asked the document authors to list the top 3 roles for which their documents were mostly intended. By and large, their answers correspond to the process shown in Fig. 2. For example, the software architect listed himself, the lead developer and the developer as intended audience for the software architecture description, and the support role as the primary target for the technical architecture description (followed by developer and lead developer).

Table 1. Participants and documents

Participant	Role	Documents authored
A	Application developer	
B	Information architect	*Vision document*
C	Lead developer	
D	Application developer + support	
E	Project manager	
F	Application designer	*Vision document for release 2* *Use case specifications*
G	Tester/testcoordinator	*Master test plan* *Test charters*
H	Software architect	*Software architecture* *Technical architecture* *Installation manual*
		Infosec/risk assessment *Proposition for release 2* *User manual* *Glossary*

Table 1 summarizes the participants and documents that took part in the experiment. The roles listed are the job titles used by the participants themselves. Some of them differ slightly from the generic overview in Fig. 2. Where participants authored certain documents, those documents have been listed in their rows. Documents for which the author did not participate are listed at the bottom of the table.

6 Experiment and Results

In this section, we present the results for each of the steps of our methodology.

6.1 Personal Construct Elicitation

In eight separate interview sessions, we elicited personal constructs for each of the eight development team members who took part in our research. Each session lasted for approximately two hours, and at each session there were two attendees: the development team member and the interviewer conducting the experiment. All interviews were conducted by the same interviewer.

Before the start of the experiment, the participants were asked to briefly introduce themselves and to describe their role in the project. During the experiment, no direct questions about (the use of) the documentation were posed; all data about the documents was elicited through triads (cf. § 2). No constructs were provided to the participants in advance, and the interviewer took care not to provide or suggest any constructs during the interview; all constructs were in the course of the experiment proposed by the participants themselves.

Table 2. Rank order grid for one team member

1	*Infosec/risk assessment*	*Proposition for release 2*	*Vision document*	*Vision document for release 2*	*Test charters*	*Use case specifications*	*Master test plan*	*Software architecture*	*Technical architecture*	*Glossary*	*Installation manual*	*User manual*	5
earlier in project	1	3	2	3	4	4	4	4	4	4	5	5	later in project
execution	5	5	4	4	2	3	3	3	2	4	1	1	support
customer requirements	2	1	2	2	4	3	3	4	4	3	5	3	realisation
has to do with testing	5	5	5	5	2	1	1	5	5	3	5	2	has nothing to do with testing
time dependency	1	1	1	1	1	1	1	2	1	5	1	1	no time dependency
understood by user	4	4	4	4	1	2	4	5	5	1	5	1	not understood by user
functionality	2	2	2	2	1	1	2	4	5	2	5	1	technique
has to do with planning	5	1	1	1	5	1	1	3	5	5	5	5	has nothing to do with planning
has to do with functionality	4	3	3	3	1	1	3	3	5	4	5	1	has nothing to do with functionality
expectation	2	1	3	3	5	4	3	4	5	5	5	5	reality
concrete (for builders)	5	5	5	5	2	2	5	2	2	2	1	2	abstract (for builders)
says something about the application	4	4	4	4	1	4	4	3	3	4	3	1	says nothing about the application
has to do with infrastructure	3	3	3	3	5	5	4	2	1	5	1	5	has nothing to do with infrastructure

We obtained eight rank order grids, one for each participant. Elicitation of personal constructs lasted until the participants indicated they could think of no more additional constructs different enough from the ones already provided. All participants provided 10 to 15 constructs before the elicitation was completed. Due to space limitations, we cannot show all eight rank order grids here. Instead, Table 2 shows a single grid to illustrate the type of results we obtained.

6.2 Shared Tacit Understanding

From the rank order grids of the eight team members, the perceived similarities between the documents can be calculated with the Focus algorithm. Documents that are similarly ranked across the constructs in a grid are more alike than documents that are ranked differently. This is illustrated in Fig. 3, which shows a hierarchical clustering of documents based on the data from Table 2. The numbers are the percental similarity scores calculated by Focus, e.g., the perceived similarity between the *Installation manual* and the *Technical architecture* is 92% and the similarity of that cluster with the *Software architecture* is 83%.

The perceived similarities between documents represent the individual's mental model of the software documentation. Table 3 shows the correlation between individuals' mental models of product documentation, i.e. the document similarity matrices. The correlations have been calculated using a simple Mantel test with 10000 randomizations.

From Table 3 we see that almost all correlations found are significant (low p-value). The only non-significant correlation is between A and E, which suggests that those two people do not share a tacit understanding of the documentation at all. Approximately half of the remaining, significant correlations are high ($\geq$ 0.5) and indicate a strong shared tacit understanding between the individuals involved. Some members of the development team (e.g., H) seem to share their understanding with most other team members, whereas some other people (e.g., B and E)

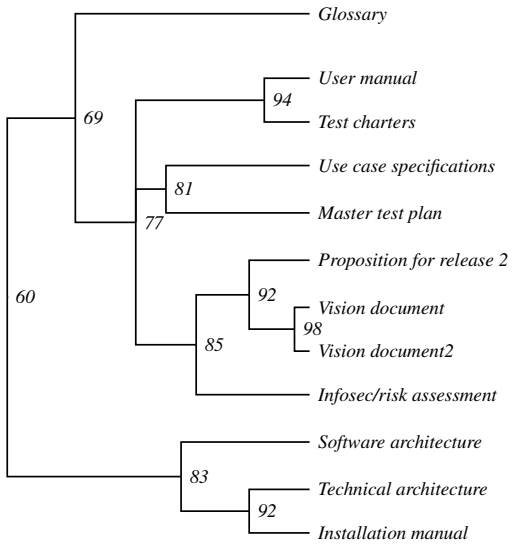

Figure 3. Hierarchical clustering of documents based on data from Table 2

Table 3. Shared tacit understanding ($p \times 10^{-3}$)

		A	B	C	D	E	F	G
B	r	0.395						
	p	2.700						
C	r	0.657	0.421					
	p	0.100	5.599					
D	r	0.541	0.486	0.600				
	p	0.300	2.200	0.300				
E	r	0.151	0.408	0.477	0.400			
	p	123.6	2.800	0.300	7.399			
F	r	0.337	0.641	0.602	0.498	0.430		
	p	4.500	0.100	0.300	2.300	2.300		
G	r	0.558	0.470	0.670	0.545	0.422	0.561	
	p	0.100	2.100	0.100	0.300	2.300	0.100	
H	r	0.472	0.440	0.703	0.519	0.601	0.501	0.755
	p	0.400	2.500	0.100	1.900	0.100	0.800	0.100

strongly share an understanding with only one other team member. There is no-one whose tacit understanding is completely detached from the rest of the team, however.

6.3 Shared Explicit Understanding

Whereas in the previous section we analyzed the shared mutual tacit understandings between development team members, in this section we will look at the explicit understanding that is shared between individuals. In Table 4 we show a summary of the team members who have corresponding constructs at a similarity of 0.80 or higher.

Table 4 shows the constructs that team members use in a similar fashion. For example, the way in which G applies the 'concept' vs. 'final' construct resembles the way in which H applies the 'realisation' vs. 'customer requirements' construct. In other words, documents that G regards as 'final' tend to be regarded by H as containing customer

Table 4. Shared explicit understanding

Team members	Corresponding constructs
A ↔ C	before start of project/end of project begin/end
A ↔ G	written earlier/later formal/informal language
B ↔ D	early/late in project what/how
B ↔ F	quality/customer domain unimportant/important (for me)
B ↔ G	solution domain/problem domain concrete/abstract
C ↔ H	begin/end supporting/execution
D ↔ F	how the software will be/has been made about type of app/about app as made
D ↔ F	first project phase/last project phase about type of app/ about app as made
F ↔ H	about type of app/about app as made abstract/concrete
F ↔ H	- functionality / technique
G ↔ H	concept/final realisation/customer requirements

Table 5. Shared understanding (tacit + explicit)

	A	B	C	D	E	F	G
B							
C	t+e						
D	t	e	t				
E							
F		t+e	t	e			
G	t+e	e	t	t		t	
H			t+e	t	t	t+e	t+e

requirements. From this table we can also see that F and D as well as F and H share more than one way of applying constructs; for all others correspondence is limited to a single pair of constructs. F and H also account for the only instance of consensus: they were the only ones to independently come up with literally the same construct (functionality vs. technique) and to apply it similarly too.

6.4 Shared Understanding

If we combine the data from Tables 3 and 4, we can identify which team members share both a tacit and an explicit understanding. Table 5 depicts this combination, and shows which team members share a tacit understanding(t), explicit understanding (e), or both (t+e). Based on the data from Table 5, Fig. 4 shows a graph-like structure in which team members that have a shared understanding (t+e) are connected. For the sake of clarity, the roles of the team members and the documents they authored have been added to the figure (cf. Table 1).

Fig. 4 seems to suggest that there is a shift in understanding of the product documentation within the development team. A chain of people $A \leftrightarrow B \leftrightarrow C$ is formed in which

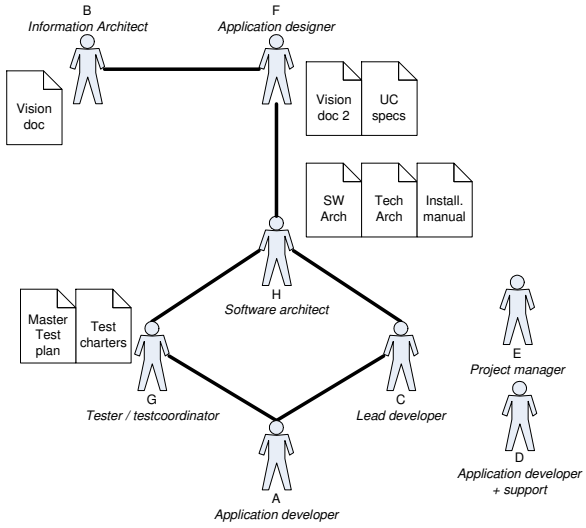

Figure 4. Shared understanding

one person B may have a shared understanding with persons A and C, whereas between A and C such a shared understanding does not exist. In a way, the figure resembles a familiar children's game known as 'Chinese whispers'. In this game, a sentence is whispered in the ear of the first child, who whispers it to another child, and so on, until the last child is reached who says the sentence out loud. Hardly ever will this final sentence be the same sentence that was given to the first child, since the message that was passed suffers from accumulated errors and distortions.

This chain-like structure of shared understanding obviously has repercussions for the way in which people's expectations of documentation do or do not match the intentions of the authors in the chain. If we look again at Fig. 4, we see that the author of the *Vision document* (B) has a shared tacit and explicit understanding with F only. This implies that his intentions for the *Vision document* are really only fully understood by F. The further down the chain, the less likely it is that people fully appreciate this document. A central role in this picture is for H, which suggests that other team members' expectations regarding the *Software architecture*, *Technical architecture*, and *Installation manual* are probably most aligned to the intentions of their author. The figure also suggests that, for whatever reason, E and D may experience the most trouble when exploring the documentation since both of them share no tacit *and* explicit understanding with any of the other participants in our experiment (although they do have a shared tacit *or* explicit understanding with various team members). Note, however, that a lack of shared understanding does not imply that team members cannot or do not read the information in the documentation. Rather, the documents' contents may be placed and interpreted in a context different than the one the author envisioned – for example, a document may be seen by the

reader as providing guidelines rather than prescriptions as the original author intended.

7 Discussion and Conclusion

One of the most striking observations from the structure in Fig. 4 is its obvious resemblance of the process in Fig. 2. People that are close to each other in the development process – because one of them uses a document that was produced by the other – also tend to have a shared understanding of the software documentation[2]. Although not a perfect match, the process followed in which high level business information is stepwise refined into more technical information is highly visible in Fig. 4.

We should reiterate here that the derivation of Fig. 4 is completely detached from the derivation of Fig. 2. In the latter case, we directly asked for information about the development process, while the structure of shared understanding in Fig. 4 has been fully based on participants' statements about the software documentation. That we were able to reconstruct an approximation of the development process based on statements solely about the documentation is a remarkable (and unexpected) result.

We understand that our findings should be further tested and verified. We know for example that the team we observed has had some issues related to documentation, especially the tendency to go back to the customer for clarifications. Although we can explain this behavior with the results we obtained, we cannot be completely sure that our explanation is the correct one. We would also like to know whether our method obtains similar results in different organizations.

Nevertheless, there seems to be an important lesson in our findings: it appears that the development process one follows is reflected in the way in which individual team members understand each other. This has obvious implications for areas such as process management and process improvement. A long, chain-like process for instance may lead to knowledge dissipation more easily than a compact, web-like structure. Consequently, if one wants to enhance a team's common frame of reference, the way in which the development process supports collaboration between individuals could be key.

Acknowledgment

This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge.

[2]Note that, even though participant F uses the job title 'application designer', his role seems to have been more that of an analyst and requirements specifier evidenced by the documents he authored.

References

[1] N. Baddoo and T. Hall. Practitioner Roles in Software Process Improvement: An Analysis using Grid Technique. *Software Process improvement and Practice*, 7:17–31, 2002.

[2] E. Bonnet and Y. V. d. Peer. zt: A Software Tool for Simple and Partial Mantel Tests. *Journal of Statistical Software*, 7(10), 2002.

[3] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, second edition, 1988.

[4] F. Fransella and D. Bannister. *A Manual for Repertory Grid Technique*. Academic Press, 1977.

[5] D. Jankowicz and L. Thomas. An Algorithm for the Cluster Analysis of Repertory Grids in Human Resource Development. *Personnel Review*, 11(4):15–22, 1982.

[6] E. Karapanos and J.-B. Martens. Characterizing the Diversity in Users' Perceptions. In *INTERACT*, volume 4662 of *LNCS*, pages 515–518. Springer, 2007.

[7] G. A. Kelly. *The Psychology of Personal Constructs*. Norton, New York, 1955.

[8] R. Klimoski and S. Mohammed. Team Mental Model: Construct or Metaphor? *Journal of Management*, 20(2):403–437, 1994.

[9] L. L. Levesque, J. M. Wilson, and D. R. Wholey. Cognitive divergence and shared mental models in software development project teams. *Journal of Organizational Behavior*, 22:135–144, 2001.

[10] S. Michelle Young, H. M. Edwards, S. McDonald, and J. Barrie Thompson. Personality Characteristics in an XP Team: A Repertory Grid Study. In *Workshop on Human and Social Factors of Software Engineering*, pages 1–7, St. Louis, Missouri, 2005.

[11] N. Niu and S. Easterbrook. So, You Think You Know Others' Goals? *IEEE Software*, 24(2):53–61, 2007.

[12] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company : How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, 1995.

[13] M. L. G. Shaw and B. R. Gaines. Comparing conceptual structures: consensus, conflict, correspondence and contrast. *Knowledge Acquisition*, 1(4):341–363, 1989.