A Cognitive Perspective on Developer Comprehension of Software Design Documentation

Hugo H. Schoonewille\textsuperscript{1} \hschoone@liacs.nl Werner Heijstek\textsuperscript{1} \heijstek@liacs.nl Michel R.V. Chaudron\textsuperscript{1} \chaudron@liacs.nl Thomas Kühne\textsuperscript{2} \thomas.kuehne@ecs.vuw.ac.nz

\textsuperscript{1}Leiden Institute of Advanced Computer Science Leiden University, Niels Bohrweg 1, 2333 CA Leiden, the Netherlands
\textsuperscript{2}School of Engineering and Computer Science Victoria University of Wellington, P. O. Box 600, Wellington 6140, New Zealand

ABSTRACT

Software design documentation is an important aid for communication during software development and maintenance. Nevertheless, little empirical evidence exists regarding the use of software documentation, and effective software design representation in particular. In an experimental setting, we used documentation from industry in which aspects of a software design were modeled in both a (UML) diagram and text. We recorded and analysed how participants used these media to answer various design-related questions and collected additional information in various questionnaires. By having participants think aloud, we set out to understand the underlying cognitive processes of developer design comprehension by applying the grounded theory method. We validated the results with concepts from the cognitive theory of multimedia learning. Results show a positive correlation between developer certainty and correctness of answers, whereas the opposite was not found. Also, self-rated experience and self-rated skill coincide with higher levels of certainty. We found that participants rated information based on perceived importance and that their “common sense” plays a significant role. Surprisingly, more than 60 percent of the answers were based on the consultation of a single medium. These results clearly ask for further investigation. We propose corresponding future work.

Categories and Subject Descriptors

D.2.7 \textit{[Software Engineering]}: Distribution, Maintenance, and Enhancement—Documentation; D.2.11 \textit{[Software Architectures]}: Languages—description; D.2.10 \textit{[Design]}: Representation

General Terms

Documentation, Design, Experimentation, Human Factors

1. INTRODUCTION

Unambiguous communication is always beneficial during software development \cite{12, 19} but of even greater importance in software maintenance and Global Software Development (GSD), when physically separated teams work on a project \cite{15}. Due to the limited possibilities for direct communication in these situations, documentation plays a more fundamental role. The degree to which Software Architecture Documentation (SAD) is intelligible can then be crucial to the success of the development effort. Software design documentation is only effective if it allows designers and developers to understand the intended design \cite{28}. Supporting this process of understanding by tailoring the methods used to convey the design in a way that fits human cognitive processing \cite{23} could increase the efficiency of communication—and thus benefit the development process \cite{1, 29}. As Mayer \cite{27} expounds, we should not perceive “understanding” as passive information acquisition but rather as active knowledge construction. In order to ease this knowledge construction, the \textit{integration} of information communicated through multiple media should be facilitated to build up a mental model. We argue that a more complete mental model decreases error and mistake sensitivity, increases development speed and reduces the amount of training a developer requires \cite{7}. Supporting the construction of comprehensive and accurate mental models is thus of great importance. In this work, we aim to develop an understanding of the development of these mental models at a fundamental level \cite{22}. In particular, this research aims at obtaining a better understanding of the relationship between human cognition and methods of documentation. Therefore, our research question is:

\textit{How do software developers comprehend software architecture representations?}

The structure of this paper is as follows: Sections two and three elaborate on the background and methodology of the study. In sections four and five, results are presented and discussed. Finally, sections six and seven outline our conclusions and future work.
2. BACKGROUND

2.1 Communication

Development of software highly depends on collaboration, which raises the potential issue of coordinating activity hinders (20). The coinciding increase in stress on communication is diverted through the informal and formal channels (20). The combination of communication tools and proper modularisation ameliorates the quality of informal communication and (or) decreases its necessity (16, 33). Still, in certain settings such as GSD or maintenance, more formal methods of documented communication constitute an indispensable complement to informal communication (6, 25). In practice, it appears that developers in general avoid using design documents when possible (20, 16, 30). In addition, in a GSD setting, developers have no choice but to use documentation, as direct communication to certain team members is hindered by temporal, geographical and socio-cultural distances (16, 24, 36). This implies that some problems inherent to GSD could be mitigated by improving documentation quality. The utility of SAD depends on two documentation properties: what the author embeds in it and what the developer extracts from it. Ideally, the latter would coincide with the former, but various factors influence the efficacy of this process. Four factors could be focused on to improve this:

- **Personal and Organisational**: It is acknowledged that people’s background influences the perceived content (21). Important to these backgrounds are the organisational aspects constituted by common ground, a shared knowledge base and shared technologies (6, 32).
- **Technological**: Technological aids for communication in Software Development. Media choices belong to this area.
- **Process**: Development processes and methods (like RUP, XP, Scrum etc.) facilitate software development. They impact on the type and amount of documentation required.
- **Representation**: Documentation may or may not fit human cognition. Modeling styles or notational conventions (28) may play a role on top of fundamental paradigm choices.

The research presented in this paper focuses primarily on the last point.

2.2 Human cognition

Both the visual and textual representations of a system should befit human cognitive processes to ensure correct understanding. This section construes a fundament for basic understanding of the human cognition.

2.2.1 Human Memory & Understanding

The path from stimulus acquisition to cognitive understanding can be divided into the processes of selecting, processing, storing, (or encoding), and retrieving information. Research into the cognitive foundations of these processes has resulted in several models of which Baddeley and Hitch’s appeared to be best able to explain a wide variety of phenomena (2, 3, 13). As can be seen in Figure 2, Baddeley and Hitch’s model subdivides human memory into three major components. The Working Memory (WM) is most interesting in software development context because this system is used to keep information active while performing complex tasks such as reasoning and comprehension (3). In here, the mental model resides, comprising attended information, available for consultation during cognitive tasks. The WM can be divided into several subsystems, depicted in Figure 1b. The total capacity of WM is limited, but each subsystem could be loaded more or less independently (2, 3). Storing information from WM into Long Term Memory (LTM) occurs through encoding. During encoding, information is organised and structured into a schema (or knowledge construct) and linked to existing schemata residing in LTM. Comprehension occurs as a result of selecting the right information and combining it with apt existing knowledge (18).

2.2.2 Cognitive Theory of Multimedia Learning

The basis of the cognitive theory of multimedia learning is the idea that the design of multimedia messages should be consistent with the way people process information. A hypothesis is that learners are able to create a better, more complete mental model using multiple types of media. Such an enhanced model in turn enables them to obtain a deeper understanding of the presented material. Three principles from the cognitive sciences found this assertion (27):

- the Dual-Channel assumption;
- the Limited Capacity assumption;
- the Active Learning assumption.

The Dual-Channel assumption suggests that it is advantageous to simultaneously use multiple channels for communicating information: the Limited Capacity assumption asserts that there is a limitation on the amount of information...
humans can process per channel at a time; and the Active Learning assumption supposes the idea that humans need to actively construct knowledge rather than just absorb presented material. The Active Learning assumption is subdivided into three separate processes:

- Selection;
- Organisation;
- Integration.

These processes are represented by the annotated arrows in Figure 1a and all take place in the working memory. Information enters through the eyes or ears into the sensory memory. Then, one selects interesting information out of all stimuli. Subsequently, the learner constructs separated verbal- and pictorial models and finally, the models are integrated into one mental model. It hence becomes clear that the integration of verbal and pictorial models is indispensable for attaining understanding. However, it is important to minimise the cognitive load during these processes.

Mayer proposed several guidelines for instructional design to improve the transfer of information and construction of knowledge by decreasing unnecessary cognitive load [27].

3. METHODOLOGY

The following subsections describe the experimental setup used to obtain knowledge about the way developers use software documentation. Also, the methods of analysis are described here. This experiment is also reported on by Heijstek et al. [14].

3.1 Sample

We applied quota sampling [37] to obtain at least one professional developer for every three students who participated in the experiment. For both groups, we applied convenience sampling. In Wellington, students participated voluntarily and went into a draw for a prize of NZS$50. In Leiden students participated both voluntarily and through a mandatory part of a course – most students were more than happy to participate, though. Industrial organisations in Wellington participated voluntarily. All students, but not all professionals, had used the UML during their studies. All participants had previous experience with the UML. A total of 47 subjects participated in the original experiment of which 35 were male and 12 female. The average age was 27, ranging between 21 and 41. The participants came from two Universities and four different organisations. Participants include BSc., BSc-hons., MSc. and Ph.D. students at the School of Engineering and Computer Science (ECS) at Victoria University Wellington (VUW), MSc. students and Ph.D. candidates at the Leiden Institute of Advanced Computer Science (LIACS) at Leiden University (UL) and developers from the field of custom software development at various different organisations in New Zealand and the Netherlands, including Capgemini the Netherlands, Infopros and ASR Insurances.

From the above described pool, 11 of the most articulate participants were drawn to ensure that enough relevant data could be collected.

3.2 Data Collection

Participants were presented with a series of different software systems. Each description comprised of a text and a diagram – separated on different sheets of paper respectively. We defined a set $T$ that holds all design information described in the text sheet and a set $D$ that holds all design information described in the diagram sheet. All design information is then described by $T \cup D$. We designed the questions so that the answer to some questions can be found in the intersection $T \cap D$, some answers can only be found in the complement $T \setminus D$ and some answers can only be found in the complement $D \setminus T$. Finally, some questions are not to be found in either set ($\sim (T \cup D)$). Dividing the two types of information (see Figure 3) enabled us to obtain information regarding media use.

3.3 Material Design

We used five text-diagram pairs that were inspired by authentic industrial SADs. We focused on the ability of participants to extract design information from both grammatically and syntactically correct diagrams and texts. All diagrams represented structural views of a system. We used UML 2 component and deployment diagrams. An example diagram is shown in Figure 3a.

Texts consisted of a natural language description of the architectural component. An example is shown in Figure 3b. Texts were designed to be in concordance with texts found in industrial SADs as much as possible. The non-verbose versions contained fewer details regarding the architectural design and were about half as long as the respective verbose versions. Questions and design documents were carefully tuned to each other to allow a multitude of subsequent analyses to be performed. Examples of questions are: “Is component X the only component that may modify attribute Y?” and “Through what node does system X connect to system Y?”. When designing questions, we limited ourselves to questions that could be answered with information that could be described in text or diagrams. We made sure not to rely on detailed knowledge of UML notation/semantics. The experiment design process involved determining and listing the most important information conveyed in the design, creating a set of questions relating to this information and validating the questions by means of trials. For each architecture two descriptions were created. One version contained a verbose diagram and non-verbose (content reduced) text and the second version contained verbose text and a non-verbose diagram. We refer to the former as a diagram-dominant representation and the latter as a text-dominant representation. We created non-verbose versions of diagrams and text by removing elements or sentences from the more complete ones.

3.4 Experiment execution

Each participant was presented two diagram-dominant representations and two text-dominant representations. The location of the verbose diagrams was alternating between left and right. The questions were the same for diagram-and text-dominant representations of the architecture. The participants were asked to think-aloud while problem solving and were encouraged to keep verbalising when they fell silent, all following the rules of Think out Loud (ToL) from Protocol Analysis [8, 9]. For further analysis of their behaviour, audio and video were recorded. In addition to the architecture representation documents we devised two questionnaires: a pre-questionnaire to collect the participant’s background information, and a questionnaire following the experiment to see how they evaluated the experiment: how they think they used the documentation and what their thoughts were about its utility.
3.5 Analysis

We used the videos and audio recordings to analyse how subjects used the architecture representations while trying to answer questions.

3.5.1 Grounded Theory: Data-driven coding

Grounded Theory (GT) was used during analysis. GT is an inductive form of qualitative research that claims to construct theories that remain grounded in observations rather than being generated in the abstract [10, 11, 26]. With GT, one does not establish hypotheses prior to performing research; the researcher creates them during analysis. With GT, one first transcribes the data and divides it into segments containing one answer per unit. Second, the process of coding commences. To remain open to every (unexpected) pattern of behaviour, no preconceived coding scheme (data-driven coding) is applied. Third, the emerged codes are categorised according to their content and relations. The final stage of theory or hypothesis construction bases itself on the outcomes of the previous phases, combined with the concurrently produced memos containing rudimentary conceptualisations and ideas that came to mind while analysing. The result of the whole process, from enquiry to writing, is a theory that theorises about the meaning of actions and relations between them [5].

4. RESULTS

During the coding stage, 20 different codes emerged, which could roughly be divided into two separate categories: behaviour and attitude. The first category captures the actual behaviour expressed by the participant, whereas the attitude-category captures the “attitude” with which the participant gave his answer. The latter is most interesting in this analysis, largely because the behaviour-codes can be assessed quantitatively as well, as has been shown by Heijstek et al. [14]. Four of the codes – all from the attitude category – will be discussed in this section, alongside with examples. They were chosen based on value and evidence available.

4.1 Codes 1 & 2: “Certain” and “Uncertain”

The first codes explored are uncertain and certain, referring to the attitude of a participant towards an architecture and related questions. A typical example of an uncertain answer is the one given by participant 29 to question 53 (“Is the communication between system x and system y secure?”):

“The only thing I see is SQL, and that doesn’t give me anything about security. So I would say it’s not secure.”

Answers coded as certainty were concise and given with confidence. The codes are especially valuable because a developer’s assessment of their own answer might be indicative for its correctness and/or developer skill. Any conclusive results could be used to be alerted of potentially incorrect answers (e.g., in the form of misreadings) in an early stage and deal with them proactively. A first hypothesis that comes to mind is a positive relation between certainty and correctness. As can be seen in Figure 4, the majority of the answers is correct, regardless of the level of certainty. However, over 70 percent of the answers coded as certain are correct - almost 20 percent more than answers coded as uncertain. However, the correlation between certainty and correctness is not significant. Some participants displayed a high propensity in being either certain or uncertain. Figure 5 shows this clearly for participant 16 and 33. Note, however, that their correctness score was not above average. In contrast, consider participant 19: 10 of his 13 answers were coded as uncertain of which only five were correct. In general, a balanced self-assessment of a developer regarding their certainty coincides with better answers albeit not statistically significant. Combining the above with the self-rated experience, skills and education obtained from the questionnaire yields Figure 6. It shows a (non-significant) correlation between higher experience and certainty.

4.2 Code 3: “Common Sense”

The third code that emerged was common sense. This code is useful to understand a participant’s reaction to am-
biguous or absent information. Does a developer acknowledge that they cannot make a decision, or do they use common sense to fill in absent information? A first observation is that participants make assumptions of the importance of information. When they think their own knowledge surpasses the presented information, they tend to prefer their own common sense: some developers are happy to base their decision on what essentially is speculation. An incorrect form of reasoning is demonstrated in the following answer given by participant 21 to question $\beta_3$ (“Can message type $x$ be ignored by system $y$?”):

“There’s no special thing about this one. Messages can be ignored is a general [sic] (...) It’s unclear about this one so I guess yes, they can be ignored.”

He probably assumes that allowing to ignore a message is the safest option, but what if that is not the case? Of course the developer’s assessment may be correct in this or another particular case, but in general they should not base decisions on assumptions. Of all participants, 4 answered question $\beta_3$ in a way we categorised as using common sense. This relatively high percentage can probably be explained with the type of question. We classified question $\beta_3$ as difficult: it cannot be answered with the information provided and we furthermore included as “false friend” – a property that could be expected to mislead someone. Many developers assume familiarity with system $\beta$ – a booking system for flights – and hence feel confident to fill in specificities using common sense. Except for participant 16’s answer on the first (example) question, every other common sense answer was coded uncertain as well. So, at least developers acknowledged that their answers were based on some level of speculation.

4.3 Code 4: “Single Medium Based Answer”

The last code analysed was Single Medium Based Answer (SMBA). Answers were coded as SMBA when the participant:

1. only looks into one of the two media and comes up with an answer,
2. explicitly mentioned that his answer was based on a single medium or
3. only peeks shortly (presumably too short to see anything valuable) into another medium.

This code is useful for understanding how developers make use of information provided in both textual and visual media and, in particular for understanding whether there is a correlation between correctness and use of media. If we only take into account (1), i.e., where the participant looked (and thus disregard the think aloud analysis), we find that the use of a single medium has a negative impact on the amount of correct answers given ($\tau = -0.279, p = 0.02$). For our wider definition of use of a single medium (SMBA), we find that 64 percent all given answers belong to this group. Of this total, 57 percent was correct (Figure 7) compared to 74 percent correctness for answers given based on both media. The use of both representations is indeed associated with the amount of correct answers. We can therefore assert that the participants seemed to have gotten a better
understanding of the presented architecture when both documents were used. This finding is in line with what could be expected based on the cognitive theory of multimedia learning.

Combining the results from section 4.1 with this code yields Figure 8. Participants answering less than nine answers using SMBA are more often uncertain. In addition, they were more often incorrect than average. These observations might be explained by the fact that participants unsure about their current answer are likely to consult both diagram and text in order to look for further evidence.

Figure 8: Certainty and uncertainty. The left five participants are constituted by the participants in the ‘less than nine answers SMBA’ group; the right six in the category of ‘nine or more answers SMBA’

Another interesting observation emerges when taking the questions as viewpoint rather than the participants. Questions $\gamma 1$ through $\gamma 3$ are all coded SMBA one or two times more often than average; questions $\alpha 3$, $\delta 1$, and $\delta 2$ are coded SMBA one time less than average. Combining this observation with the correctness of the answers on these groups of questions in Figure 9, we see that answers given in the ‘less than average’-category are correct more often. Questions that make a participant more susceptible to answer it using only one medium, increase the amount of errors as well. The present research however cannot provide conclusive thoughts regarding this distinction.

Figure 9: Results of questions coded SMBA. The category ‘less than average’ is constituted by questions $\alpha 3$, $\delta 1$ & $\delta 2$; ‘greater than average’ contains questions $\gamma 1$, $\gamma 2$ & $\gamma 3$. ‘Total’ consists of only answers coded SMBA. The numbers represent the actual amounts

5. DISCUSSION

5.1 Certain and Uncertain

Holmes identified that second language learners could experience difficulty in expressing modal meanings [17]. As most of our participants were non-native English speakers, their expression of their understanding of the documentation could be distorted. In turn, this could result in the absence of a clear relation between certainty and (in)correctness. The positive correlation between certainty and correctness might be explained by participants taking educated guesses based on scarce cues combined with experience and skills that turn out to be correct. Our results indicate that a propensity towards extreme positions regarding the self-assessment of one’s abilities is undesirable. High overall uncertainty might indicate that the developer is less knowledgeable; high overall certainty might indicate that the developer has an overrated self-image regarding their work. Any such low correlation between the perceived abilities and the actual abilities is potentially problematic. In our experiment, balanced self-assessments correlated with better quality answers. Another useful observation we made is the positive correlation between the level of experience and correctness of answers.

5.2 Common Sense

In general, the ability of developers to speculate about the importance of information is not bad in itself. Experience and skill may often lead to correct assumptions. For example, developers sometimes implicitly use the Level of Detail in a UML diagram to successfully assess the importance of a component [31]. However, developers sometimes err in their judgment, as partly confirmed by our experiment. They could be helped by emphasising important parts of the system in the textual and pictorial description, hence reducing extraneous processing. To avoid developers being forced to fill in missing information more or less successfully, the authors should be aware of the guidelines proposed in the cognitive theory of multimedia learning. Documentation should be concise so developers can actually find the relevant information and are willing to check the whole documentation [34]. It should be designed to reduce extraneous and manage essential processing by facilitating integration of both textual and pictorial representations. Besides, it should also be up-to-date so it contains the correct information. Ideally, knowledgeable colleagues can be consulted by a developer but this is not always possible, in particular in offshore development scenarios. This experiment simulated such a situation. Gaps in documentation are less problematic if the author and recipient share some common ground. Common ground refers to the knowledge that different people have in common and that they are aware of the fact they do [32]. Common ground can thus decrease the chance of miscommunication: when some detail was omitted from the documentation by the designer – maybe even intentionally so – every developer sharing the same common ground is then able to fill in the correct answer. A decrease in surface completeness could hence be complemented with this common ground.

5.3 SMBA

Our observation that answers coded SMBA are slightly more often correct than average may not be generalisable
to all sorts of documents since our software designs did not contain any inconsistencies. The behaviour of participant 19, for instance, might simply be the result of a lower level of experience and hence less correct answers overall. In general, participants answering less answers based on a single medium are more uncertain. As mentioned before, a possible explanation for this could be that they more often chose to consult both media to come up with an answer while being uncertain about their current answer candidate. This in turn could be due to the absence of data in one medium, but then every participant would have been expected to show roughly the same pattern. A more likely cause therefore is an inability to extract the requested information or the desire to double check answer candidates using all available information because of a general level of uncertainty. Either way, our results clearly indicate that using multiple media in forming an answer and uncertainty are positively correlated.

Answers to a few questions were noted to be coded as SMBA one time more or one time less than average. In the ‘answers coded as SMBA less than average’ question category, more answers were correct. This apparently contradicts earlier results, which suggested that correctness increases with an increase in SMBA, but note that we are only considering the subset of SMBA answers here. Questions that make a participant susceptible to answer it using only one medium, increase the error rate. Causes for this effect might be found in the architecture at hand, but that only holds for the ‘answers coded as SMBA more than average’ category, as all these questions came with architecture γ. The ‘less than average’ category shared a different common denominator: their topological nature. Apparently, people struggled less with questions of such nature.

Most importantly, over 60 percent of the answers were coded as SMBA. This SAD usage pattern raises some major questions regarding the creation of such documentation. Is it even advantageous to use multiple media simultaneously or should the focus be on improving single-media documentation? This shows a use for the hypothesis of the cognitive theory of multimedia learning. SMBA-amounts could be decreased and performance increased by facilitating integration. The hypothesis states that a better, more complete mental model of a system to-be-build can be obtained by using both pictorial and textual models, but emphasis should be placed on ease of integration. Using the same words for the same components in both documents for example, simplifies selection and integration. Hence, both documents would be used more often and deeper understanding of the model could be reached.

6. CONCLUSION

The research presented in this paper provided a peek into the developer’s mind: we saw how extreme self-assessments can be an indicator of problematic performance and that developers are happy to fill in missing information without necessarily having the common ground with the author of the documentation that justifies such a step. Being certain about one’s findings is good, but only if the certainty is justified. Hence, even more important than being right most of the time is the ability of a developer to successfully self-assess whether a perceived certainty is justified or not. Such critical, meta-cognitive thinking is the prerequisite for stopping developers to make unjustified assumptions or incorrect interpretations but consult further sources instead. In part the research presented here shows the suitability of the Grounded Theory method to improve Software Architecture Documentation. We believe it was successful in that it produced a number of emergent codes that revealed interesting developer characteristics. Together with the guidelines proposed within the cognitive theory of multimedia learning, textual and pictorial representations could be aligned better. Using the same words for the same components, highlighting important parts and showing linked parts near each other are only a few examples of how integration of the two representations could be improved. This analysis resulted in a number of promising directions for future work.

7. FUTURE WORK

Potential future research based on this work includes investigations addressing the following directions and questions.

The tendency of developers towards being certain or uncertain, their justifications, and the effects thereof should be further examined. As proposed by Holmes [17], developer certainty could be expressed using a quantitative scale. Such a quantitative measure could subsequently be related to the effect size, e.g., to defect density.

The self-awareness of developers about their attitude, e.g., regarding filling in missing information, could be assessed. We hypothesise that increased self-awareness results in better software, as the developer is then expected to move often ask for elucidation when necessary. Results confirming this hypothesis could then be used to create self-awareness training for developers to increase their performance.

The relation between uncertain and common sense should be looked into. Does an inherent uncertainty of the developer imply an increased use of common sense or is it the other way around? In the presented research, common sense implies uncertainty in most cases, but not to a statistically relevant extend. Does a perceived familiarity with a system domain automatically imply an increased use of common sense and if so, why? Is the problem that authors (intentionally or unintentionally) leave out details they expect to be known by developers, or is it that developers are inclined to use knowledge that is most readily available to them – i.e., in his memory rather than in the documentation (an effect known as the availability bias [35])?

Further research should address the correctness and completeness of the mental model. The contents of it are usually a mixture of perceived and retrieved information from Long-Term Memory, but there is no guarantee that it is a healthy mixture. We hypothesise that increased application of common sense generally implies a mental model that excessively relies on Long-Term Memory information, hence increasing the error rate.

How do inconsistent and/or incomplete SADs affect the quality of the resulting software? The effect of inconsistent architectural documentation in software development is unknown. We hypothesise that scenarios that make developers susceptible to consulting only part of the available documentation have a detrimental effect on software quality because inconsistencies in the documentation that could have been picked up by a comprehensive consultation slip through unnoticed. In some cases, this will lead towards inconsistencies in the developed system [31]. As a result, it may not always be a good idea to distribute information across different media if it is known that some developers have an adversity to
one media type (e.g., UML) and/or are known to excessively rely on another media type (e.g., text).

Finally, thus another interesting question is: what makes developers choose only one medium? Is it too burdensome to integrate information from different sources? Are their personal preferences, such as past bad experiences with certain types of media (e.g., sub-standard UML diagrams)? Is it even advisable to use multiple media at all, or should one rather focus on improving the quality of single-media documentation?

8. ACKNOWLEDGMENTS

The authors would like to thank all participants and Craig Anslow, John Hine and Jason Low for their support during the experiment.

9. REFERENCES


