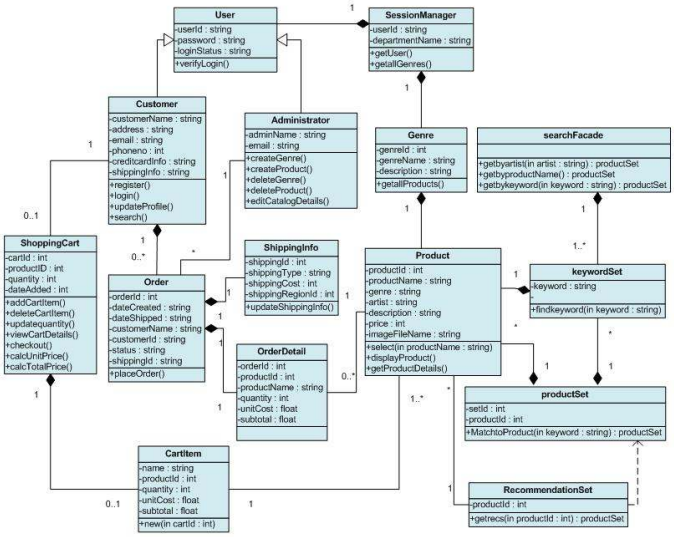
Software Architecture Documentation

A cognitive perspective



Hugo Schoonewille

Natura non facit saltum

Charles Darwin - On the origin of species (1859)

Software Architecture Documentation

A cognitive perspective

Leiden, January 17th 2011

Author:	Hugo H. Schoonewille	h.h.schoonewille@umail.leidenuniv.nl
1st Supervisor:	Werner Heijstek	heijstek@liacs.nl
2nd Supervisor:	Michel R. V. Chaudron	chaudron@liacs.nl

Abstract

How do software developers comprehend software architectures? The main question addresses several aspects inherent to software development and this research aims to reveal the effects of the behaviour displayed by developers and subsequently provide ideas for future directions in research.

Experiments were conducted using the Think Out Loud method to gain insight into the developers' minds. Experiments were recorded, transcribed and subsequently coded in order to uncover patterns of behaviour, following the qualitative method of Grounded Theory.

Results show that usage of "common sense" to supplement excluded or unclear details of a software design correlates with a participant being certain or uncertain. In addition, several participants showed a tendency towards certainty or uncertainty – regardless of their answer being correct or incorrect – and there exists a positive correlation between that and their self-rated skills and -experience in software development. A third observation was that several participants were inclined to base most of their answers on a single medium. Last but not least, unanswerable questions combined with an architecture with *false friends* increases the amount of common sense applied.

Results based on an additional analysis using the stages of the Active Processing Assumption as codes revealed that answering during integration is undesirable as more answers given during this stage were incorrect.

Combining these outcomes with extensive research into the principles of the Cognitive Theory of Multimedia Learning, Global Software Development and Object-Oriented Development, provides a solid ground for future research with a vast amount of hypotheses to be addressed in the quest for an answer to *the* great question: *How can software architecture documentation be improved?*

Table of contents

Appendices

1. Introduction

Translating a software architecture into good code corresponding to the architecture, requires a process of understanding the documentation, rendering the human element critical in software development [52]. Humans have the innate ability to integrate pictorial and verbal models (in this case diagrams and textual descriptions) in one so-called mental model. In Software Development, this mental model is constructed from Software Architecture Documents (SAD) and subsequently consulted for programming decisions and problem solving [44]. A more complete mental model decreases error and mistake sensitivity, and increases development speed at the same time. Supporting the construction of such mental models is thus of great importance and a cognitive perspective on this challenge might yield interesting implications.

2. Background & Related work

These fields of study are relevant to this work. It provides a motive for the importance of the current study but virtual tools and conceptions that enable us to perform adequate research and draw solid conclusions as well.

2.1. Documentation in Global Software Development[1]

The search for competitive advantages often forces companies to look abroad for cost reduction, quality enhancement, flexibility increase, risk dilution and productivity improvement [56]. Globally distributed software development has many potential benefits but brings lots of drawbacks as well [9, 16, 55, 8]. Moreover, the assumed benefits do not seem to materialise [20].

Either way, the benefits of Global Software Development (GSD) cannot be exploited without effective mechanisms for information- and knowledge-sharing [49, 59, 36, 34]. Research indicates that GSD imposes several problems: increased communication delay and work completion time, both due to a lack of informal communication and difficulties in finding the right expertise and knowledge [14]. Therefore, the main challenges lie in the complexity of maintaining efficient communication and coordination when teams are dispersed [40, 36].

[1] Appendix A3

Investing in innovative tools to increase cross-site awareness and ameliorate communicative impediments, and research into the workings of development methods are two ways to go, but documentation plays an important role as well. However, the unpredictable nature of development requires tools for documentation to facilitate easy changeability and means to make those modifications noticed by the involved developers as well. Furthermore, improved capture of informal, transient designs, and a decrease in effort required to keep them up-to-date mitigates some difficulties [42].

"The solution to the problems introduced by GSD is neither to add more documentation nor to abandon it: it is to get better documentation. We must go towards lean, concise documentation" [1]. The combination of a well established common ground, – the knowledge different people have in common and that they are aware of the fact they do[1] – easily accessible knowledge and concise documentation appears to fit GSD best.

2.2. Object-Oriented Development[2]

The Object-Oriented (OO) paradigm was introduced in the late 60s with Simula 67, when several events in the development of hardware, programming languages and methodologies and database models allowed for it [2, 10: 33-34]. The idea was to exploit the human mind's natural capabilities for thinking about the world in terms of objects and actions to deal with the ever-increasing complexity of software systems [27]. And indeed, research has shown that novices tend to prefer OO-techniques over procedural techniques [46]. Also, the construction of a design solution based on structures in Long Term Memory (LTM) eases its maintenance and improves robustness. Moreover, besides the cognitive advantages, improved reusability of design increases development pace, quality and flexibility [10: 74-76; 57, 51].

However, these are not merely improvements: a stumbling block to obtaining the benefits of OO is learning the approach [46]. Also, individuals may construct invalid analogies, using their prior experience inappropriately [46], or may not even possess the relevant design schemas [57]. Moreover, while objects, classes and inheritance certainly have an intuitive flavour, their formal version in OOD differs in important ways from their origins [27].

---

[1] *Common ground* is easily established when collocated, because developers then not only share cultural and linguistic backgrounds, but there is also better awareness about the microcontext – what one's doing, what remains to be done et cetera [49]. Common ground is also known as *transactive memory*.

[2] Appendix A2

In conclusion, more research is needed on why OOD does not always alleviate the difficulties encountered and what could be done to improve and ease the processes intrinsic to software development. Where Brooks thought that "[t]he most important single effort we can amount to is to develop ways to grow great designers", we think that there is much to be gained from improving the development methods and means themselves as well [11].

2.3. Cognitive psychology

2.3.1. Human memory and understanding[1]

People have to process stimuli before understanding can occur. This path can be expounded into the processes of *selecting* (or attending), *processing*, *storing* (or encoding) and *retrieving* of information. Research on the cognitive foundations of these processes resulted in several models, of which Baddeley and Hitch's is most successful [3, 4, 6]. Its success hinges to its ability to explain a wide range of emperical phenomena [29, 4]. Hence, this model will be discussed here.

As can be seen in figure 1, human memory is subdivided into three major parts of which the Working Memory (WM) is most interesting here. WM is used to keep information active while performing complex tasks such as reasoning and comprehension [6]. Here resides a *mental model* consisting of attended information, available for consults during cognitive tasks.

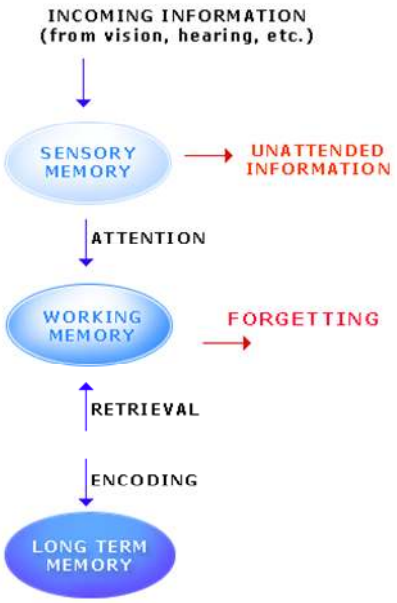


Figure 1: Human memory systems, adapted from [5: 2]

[1] Appendix A1, section 2

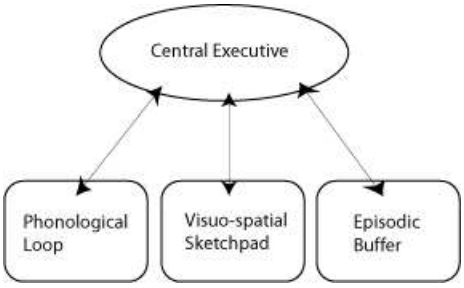


Figure 2: Model of the Working Memory, adapted from [6]

The WM can be divided into several subsystems, depicted in figure 2. The phonological loop processes auditory and verbal information; the visuo-spatial sketchpad handles the visual information. The central executive represents an all-purpose attentional controller and supervisor of its slave systems. Recently, the episodic buffer was introduced, which is assumed to facilitate integration of information in the other subsystems. However, further research is needed to gain more insight into its workings [4, 6]. The total capacity of WM is limited, but each subsystem could be loaded more or less independently. .

Storing information from WM into LTM occurs through encoding. During *encoding*, information is organised and structured into a schema (a knowledge construct) and linked to existing schemas residing in LTM.

Believed is that the organisation and integration phases (described below) account for human understanding and enables them to retrieve suitable schemas to apply when encountering new situations [44]. Understanding occurs as a result of selecting the right information and combining it with apt existing knowledge [38].

2.3.2. Cognitive Theory of Multimedia Learning [1]

At the basis of the cognitive theory of multimedia learning stands the idea that the design of multimedia messages should be consistent with the way people process information. This is based upon three principles of the cognitive sciences:

1. Dual-Channel assumption;
2. Limited Capacity assumption;
3. Active Learning assumption.

In short, the *Dual-Channel* assumption represents the separation of channels for processing visual and auditory information; the *Limited Capacity* assumption assesses the limitation on the amount of information humans can process per channel at a time; and the *Active Learning* assumption refers to the idea that humans need to construct knowledge from presented material rather than just to absorb it.

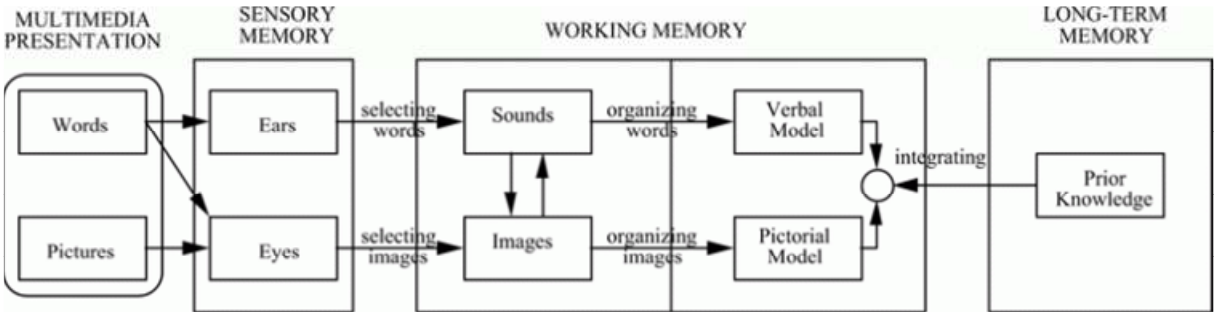


Figure 3: Cognitive Theory of Multimedia Learning [44: 61]

The integration of verbal and pictorial models is crucial in successful multimedia learning, and can be equated with the Active Learning-assumption: this assumption is again subdivided into 3 separate processes (depicted by the annotated arrows in figure 3):

1. Selecting;
2. Organising;
3. Integrating.

Information enters the pathway through the eyes or ears in the sensory memory, and subsequently the first stage of cognitive processing commences: first, one selects interesting information out of all stimuli. Second, the learner constructs separated verbal- and pictorial models and third, the models built are integrated into one mental model.

[1] Appendix A1, section 3

As described above, learning requires cognitive processing and hence induces cognitive load. Three kinds of cognitive load are to be discerned:

1. Extraneous Processing;
2. Essential Processing;
3. Generative Processing.

Extraneous Processing encloses cognitive processing that does not serve the instructional goal. *Essential Processing* is required to represent the essential material in WM and *Generative Processing* is required to create a deeper understanding of the presented materials. In order to ameliorate the process of selecting, organising and integrating, Mayer has proposed several guidelines for instructional design to improve the transfer of information and construction of knowledge: instructional designers need to reduce extraneous processing, manage essential processing and foster generative processing [44].

The assertion that people learn better from words and pictures than from words alone is built on the hypothesis that learners are able to create a better mental model using both media. This enhanced model in turn enables them to obtain a deeper understanding of the presented material. Mayer's empirical research generally endorses this stance [44].

3. Research Question

GSD introduces complexities that corrode most of the intended improvements. The majority of these new problems are brought about by new impediments in communication and coordination between different teams. Unambiguous, straightforward and quick communication is important [40, 28], but of even greater importance when physically separated teams work on one project [31]. A decrease in both the amount and effectivity of communication, imposes the software architect with the task of not only designing the structure of the software, but he has to take into account the development task dependencies as well [31]. An SAD has several fingers in this communication-pie being the foundation of many software projects. Increased intelligibility of this documentation could hence make or break the software product aimed at.

It ultimately is the software developer who has to understand what the architect wants. Unfortunately, this is not as straightforward as one might think: Object-Oriented Designs like UML use the innate intuition of using objects and actions, but this sometimes clashes with the very intuitions it produces [27]. Slight differentiation between the intuition and the formal object-model impose (unnecessary) difficulties upon each party involved.
Also, the increasingly complex SADs lead to a restricted perceived application domain, resulting in the fact that developers do not use its full potential [58], primarily in the early stages of the system development process [23, 24]. Together, this makes that inconsistencies in SADs are not noticed or simply ignored [23]. As a primary motive, many studies using Protocol Analysis examine the theoretical underpinnings of SE in order to advance the maturity of the discipline [37]. To understand how developers comprehend SADs is of vast importance as good understanding of an SAD reduces the probability of incompatibilities between the work of different development teams, hence ameliorating the software quality. With this research we want to add our own attribution to this process.

Thus, the idea behind this research is to find ways to improve documentation to fit the workings of the human cognition to a further extend. Therefore, our research question is:

How do software developers comprehend software architectures?

4. Experiment & Analysis

The better the researcher and setup, the better the data and the less participants needed. However, a lot of low quality data can never replace less high quality data [12: 230].

This section describes the experimental setup and research materials used, and provides an overview of the methods of analysis applied.

4.1. Research design

4.1.1. Experimental setup

To create a situation in which we could acquire data, we created a set of 4 different SADs, all consisting of both a textual description and a diagram based on UML. These SADs was presented to people with experience in software development and we questioned them about specific properties of the system depicted by the documentation. Using a video camera and voice recorder, their behaviour was recorded as they were posed with the different questions.

Two questionnaires were devised: a pre-questionnaire to collect background information on the participants (e.g. their current knowledge, employment and native language). A second questionnaire following the experiment was used to see how the participants evaluate the experiment: how they think they used the documentation and what their thoughts are about its utility.

Figure 4 and 5 depict the physical setup and placement of the participant. Figure 5 also shows the positions where the documentation is placed. In order to prevent biases (i.e. uncertainty with the first question and preference for left or right), the order of documentation presented changes with each participant and the position of the text and diagram with every documentation. Furthermore, per participant we used documents with different levels of verboseness: alternately the text and diagram will contain the more verbose description.

Figure 4: Experimental setup

Figure 5: Position participant

4.1.2. Participants[1]

We aimed to find participants who are currently working in the software development industries. However, taking into account the difficulty to find people who want or are able to spend half an hour on participating in our experiment, we looked for students as well. In this experiment, only a small amount of 11 subjects drawn from a pool of 47 were analysed in depth. This is mostly due to time constraints, but qualitative research in general never relies on notions of statistical representativeness.

Because the experiments already started prior to my arrival, *sampling* occurred only after finishing the experiments. This contradicts Grounded Theory as described below, but we assert that it in spite of this will result in more or less unbiased outcomes: we selected the most articulate participants; the other properties of excellent participants were satisfied anyhow[2].

4.1.3. Conduction

After the pre-questionnaire, the participants received example documentation and an accompanying question to get familiar with our expectations. The participants were asked to think out loud (ToL) during the complete experiment, a part of Protocol Analysis, as defined by Ericsson and Anders [25]. We recorded what the subjects are looking at and what they are trying to find. After the example question, 4 other SADs follow, accompanied with 3 questions each. In every execution of the experiment, we changed the order of SADs, position of the text and diagram and their verboseness following the experiment design. As a backup for the audio and video recordings, we took notes of what the participants say and do as well.

[1] Appendix B, section 4.1

[2] Excellent participants meet the following requirements [12: 231]:

- They are known with the phenomenon under investigation;
- They are willing and have the time to participate;
- They are articulate.

4.2. Approach analysis

Two separate approaches in analysis were taken. The first and main approach is discussed in sections 4.2.1 through 4.2.3. Description of the second approach can be found in section 4.2.4.

4.2.1. Methods applied[1]

The gathered data needs to be analysed to be able to derive a theory from it. The collected data in our experiment consists of video and audio fragments of subjects looking for information that could help them answering the questions. Audio was chosen to be the main source of data, as we asked the subjects to think aloud (using ToL from Protocol Analysis) while working on the questions.

Grounded Theory (GT) will be applied to analyse the data in further detail. This method is an inductive form of qualitative research that claims to construct theories that remain grounded in observations rather than being generated in the abstract [26, 43]. With GT one does not establish hypotheses prior to performing research; the researcher *creates* them after profound analysis of all data available.

4.2.2. Why these methods[2]

Although GT has proven to be a difficult approach, we unflinchingly faced this alleged complexity: because we could safely presume we do not know anything about the actual processes going on in the heads of our subjects, an open approach like that of GT seemed suitable. Besides, it is commonly used for similar studies. Furthermore, the phases of GT provide guidance to structure the data and disclose its unapparent properties. In addition, GT is a relatively new method in Software Engineering and it is acknowledged that theories can prove their relevance because the method of inquiry is uncommon in the area under investigation [17: 153].

The combination of both methods enables us to obtain qualitative data as objective as possible and it provides us with a tool to analyse this data, again in an unbiased and open manner. Therefore we assert that data gathered using ToL is pre-eminently suitable for being analysed through GT.

---

[1] Appendix B, section 1

[2] *Ibid.*

4.2.3. Constructing a theory

The construction of a theory is an intricate and complex task. First, the data needs to be **transcribed**[1] in order to get a grip onto the data. Henceforth, the transcribed data is **segmented**[2] into units of more or less context independent information. In our analysis, these segments consist of the answers to the question, each individually. Once finished, the segments need to be **coded**[3] using data-driven coding[4]. Such an emergent vocabulary suits GT very well, especially when the researcher refrains from reading literature. However, we did read literature: it is impracticable to start the analysis with a complete *tabula rasa*: especially novices with GT need a perspective to be able to distinguish relevant data from the irrelevant and to abstract significant theories.

After complete saturation[5] is achieved, the outcomes need to be **categorised**[6]. During all phases, we write notes and *memos* to capture our nascent ideas about the structure in the data, to conceptualise and produce an abstract account of it [12: 245]. Writing those memos is accounted to be the fundamental process in generating a grounded theory; transcribing, segmenting, coding and categorising are mere tools to aid this process. During categorising, codes are chosen for their importance and generality regarding the aspects under investigation. The researcher is obliged to look at relations between the codes with the aim to raise the level of concept from descriptive to analytic, ultimately to construct a better theory or hypothesis [17: 186].

Finally, the stage of **theory construction**[7] announces. During coding, categorising and its concomitant memoing, structure in the raw data became apparent. One or more core categories have emerged and memos contain nearly all important information, all data could be tied together. Memos and categories are looked over once more during the final writing and rewriting of a research thesis.

---

[1] Appendix B, section 4

[2] *Ibid.*, section 5

[3] *Ibid.*, section 6

[4] A data-driven coding vocabulary is constructed through analysis of the data: codes simply emerge from the data, hence enables the researcher to remain open and unbiased towards unexpected discoveries.

[5] *Saturation* is the process in which codes are still developing. It is the cristallisation of codes, categories or the theory. Saturation is reached when the researcher does not perceives anything new anymore [12: 117]: when saturation is reached, there is no new data that could not be accounted for by the grown hierarchy of codes, categories, and eventually the grounded theory.

[6] Appendix B, section 7

[7] *Ibid.*, section 8

The result of the whole process, from inquiry to writing, is a theory that theorises about the meaning of actions and relations between them [17: 151]. The emergent theory is vividly described and cogently underpinned with the use of the written memos accompanied by an introduction and conclusion. Besides that, extensive literature review from the field under investigation as well as other domains back up the constructed grounded theory. The result is an argument fully based in empirical observations [43].

4.2.4. A second approach: concept-driven coding

Contrary to the data-driven approach with GT, we analysed the transcript of 1 participant from another angle as well. A predefined coding scheme based on the phases of active processing (see figure 6 and section 6.1) was applied to one participant complying with the properties of an excellent participant. By using the phases of active processing as vocabulary, high reliability could be achieved. In addition, it enables for an incorporation of the results into a larger body of research due to the use of equivalent (static) terminology.

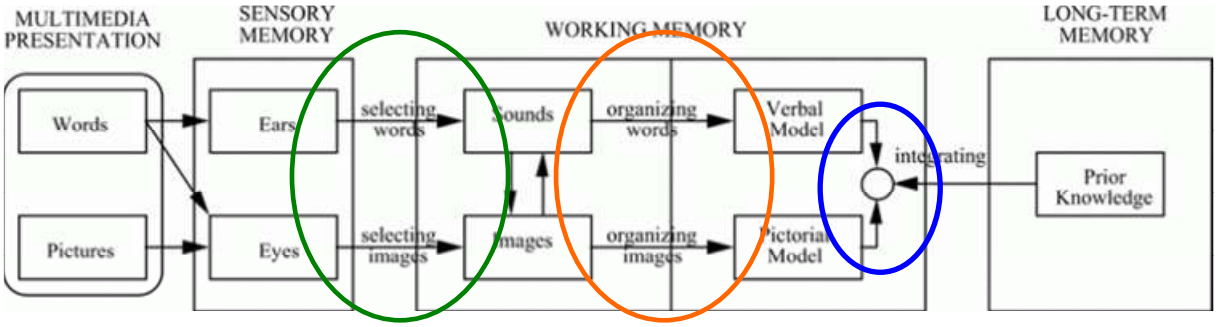


Figure 6: Phases of the Active Processing Assumption, adapted from [44: 61].

4.2.5. Results

The results will be discussed in two parts, as the analysis was twofold as well. Section 5 discusses the analysis based on GT, in section 6 the analysis based on the phases of active learning is expounded and an overall conclusion will be drawn in section 7.

5. Analysis 1: Grounded Theory

5.1. Introduction

All methods for knowledge elicitation – including the application of Grounded Theory (GT) – rely upon interpretation by the experimenter. This renders the analysis phase a crucial creative step in the development of any substantial theory [53]. Once the phases of experimenting, transcribing, and coding are finished, the moment of theory construction announces oneself.

The results presented in this section are obtained during careful analysis of the emerged codes. Writing memos to externalise thoughts and ideas, and to saturate categories, enabled us to link the emerging theories and to create an overview of the researched area.

The structure of this section is as follows: section 5.2 describes the codes obtained and will expound their underlying rationales and section 5.3 reflects upon the results and draws conclusions; Limitations can be found in section 8.

5.2. Code Acquisition & Analysis

5.2.1. Code development

The codes analysed were all obtained following the protocol of GT. Using this in combination with Think out Loud (ToL) the most objective and thorough results were acquired. Before the process of coding could commence, the data was transcribed and segmented into segments containing one answer per unit. Following GT, we chose to code these segments using a data-driven coding scheme: at the risk of a decreased reliability we wanted to remain open to every (unexpected) pattern of behaviour. To capture the behaviour to the fullest extend, allocation of multiple codes to each single segment was allowed. When the emerged codes were fully saturated, the subsequent phase of categorising was initiated. More detailed information on rationale and realisation of the above mentioned processes can be found in Appendix B.

5.2.2. Further Analysis

During the coding stage, 20 different codes emerged, which could roughly be divided into two separate categories: *behaviour* and *state* types. The first category captures the behaviour expressed by the participant. The *state* category on the other hand, captures the "state" or "attitude" in 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.* [30]. The following table depicts the emerged codes and divides them into the two categories:

Behaviour	State
1 Medium thorough	Certainty
2 Media thorough	Assuming
Component First Search	Confusion
Property First Search	Definition
Cross-media check	Technical knowledge
Rerequest question quick	Common sense
Rerequest question slow	Superfluity
Scan	Single Medium Based Answer
Term search	Remembering
Related term search	Misunderstanding

Table 1: Codes and categories

The surplus value of the "state" category made that the emphasis in the analysis will lie on the codes contained by this category. Furthermore, the analysis will be restricted to four codes of this category, chosen based on value and evidence available. The more elaborate reasons for them being picked are discussed in the accompanying introductions.

The structure in the continuation of this section will be as follows: sections 5.2.3 through 5.2.5 will describe the codes and provide empirical evidence for them. After that, relations to other codes and generalisations amongst participants will be discussed, along with probable theoretical underpinnings and resulting conclusions. In the final subsection, an overview of the conclusions drawn is given.

5.2.3. Assuming and Certainty

5.2.3.1. Introduction

Both codes are chosen based on their omnipresence in the data, which assures us of plenty evidence available. In addition, these codes are especially fascinating because it is interesting to know whether an attitude of a participant towards his own answer implies something for its correctness or not. Results could be used to notice uncertainties in an early stage and consequently deal with it.

5.2.3.1.1. Assuming

This code could be defined in the following way: "The participant clearly makes assumptions to form an answer." Roughly, the following utterances imply this code:

- I think …
- It seems …
- I guess/it's a wild guess …
- I don't know, but I choose …
- "I would say …" in combination with some assumptions or explicit limitations in searching through the documentation mentioned. (i.e. the participant mentions that he only looked into one of the media and bases the answer on that – incomplete – information.
- Well then I'm going with …

However, it excludes:

- "Yes/no, it should(n't) be a problem." (For constructions based on this, see *common sense* in the upcoming section 0).
- "I would say …" in general. See above for other cases. Verbalisations of this kind neither get this code nor the code *certainty*.

5.2.3.1.2. Certainty

Answers given confidently are coded as certain. These answers are recognisable mostly on their concise form and that they are uttered with great confidence. Other characteristics that are taken into consideration are for example utterances like "I'm sure …" or "It must …"

5.2.3.2. Examples

An example of an answer coded as *assuming* can be found in the answer of participant 29 on question Delta 3:

> "*The only thing I see is SQL, and that doesn't give me anything about security. So I would say it's not secure.*"

Participant 35 on question Delta 1 is assuming as well:

> "*And, well, the protocol doesn't say anything about what kind of information is present. Well, it does say something but … I see an attribute about billing client but it doesn't say anything about the messages. Well, it's linked to the billing server, so I think its messages about the clients the bill of the client …*"

An example of an answer coded *certainty* is the answer of participant 19 on question Gamma 2:

> "*Mortgage webserver ... No.*"

And another quotation depicting behaviour coded as *certainty* comes from participant 28 and his answer on question Alpha 1:

> "*Which external system … Paraplu.*"

5.2.3.3. Analysis

First should be noted that both codes complement each other: they never appear together with one answer. The following table contains the answers coded as *assuming* or *certainty* and whether the coded answer was correct or incorrect. The hypothesis is that most answers coded *certainty* will be correct and that *assuming* will have a higher percentage of incorrect answers.

	Correct	Incorrect	Total
Assuming	35 (56%)	27 (44%)	62
Certainty	49 (74%)	17 (26%)	66

Table 2: Answers correct or incorrect per code

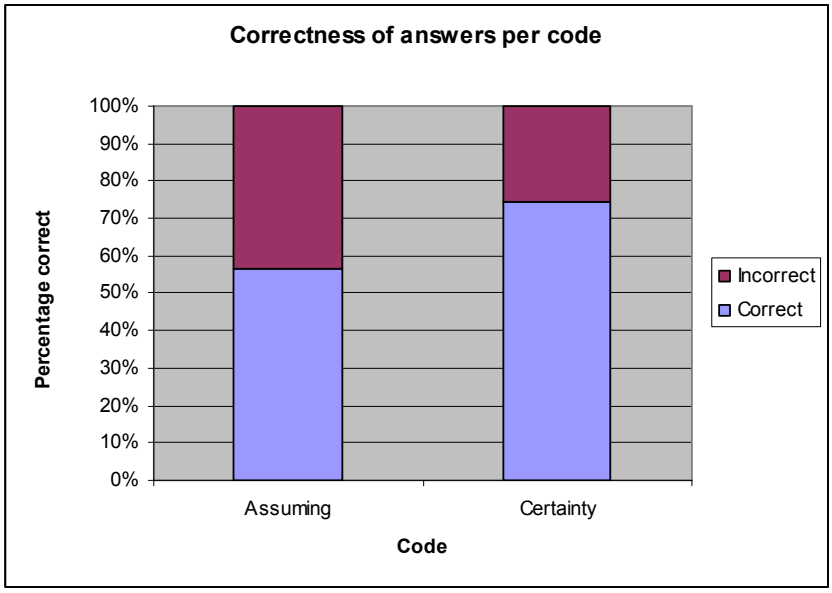


Diagram 1: Answers correct or incorrect per code

The above diagram depicts our expectations merely partially: in the case of the answers coded with *certainty*, there is a majority of answers correct; when participants based their answer on assumptions and were somewhat uncertain about their answers, they still guessed correct in somewhat over 50% of the cases.

Placing these results in the light of every individual separately, it appears that some participants have a tendency to be certain or assuming. This propensity towards one of both codes results in expected patterns of unwanted behaviour. For example, participant 16, who is overly certain about his answers (11/13 of his answers were coded with *certainty*), gave 8 correct answers of which 7 were coded *certainty*. The same holds for participant 33: he has 9 of his 13 answers coded with *certainty*, of which only 5 were correct. Diagram 2 depicts this very nicely: participants 16 and 33 stand out in their certainty, yet the percentage of correct answers does not deviate from average. The other way around there is less evidence: the most compelling case is to be found with participant 19: 10 of the 13 answers are coded as *assuming* of which only 5 were correct.

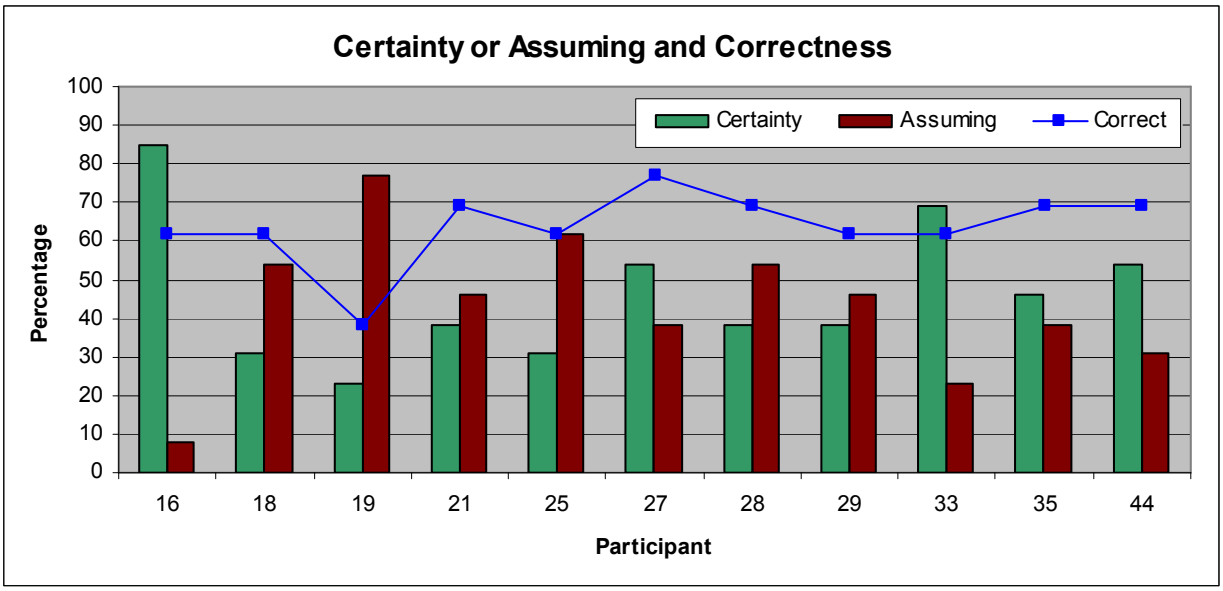


Diagram 2: Correlation between *certainty*, *assuming* and correctness

Furthermore, notable is the fact that, while participant 16 was overly certain, he rated his experience and modeling skills above average. The evidence for participant 33 however is less compelling: he rated only his modeling skill at the maximum level. On the other hand, we see that participant 19 rated those significantly below average. Diagram 3 is sorted on the amount of answers coded *certainty* to make this average trend stand out clearer. The 'mean level of experience' represents the averaged value of all self-rated skills and experience.

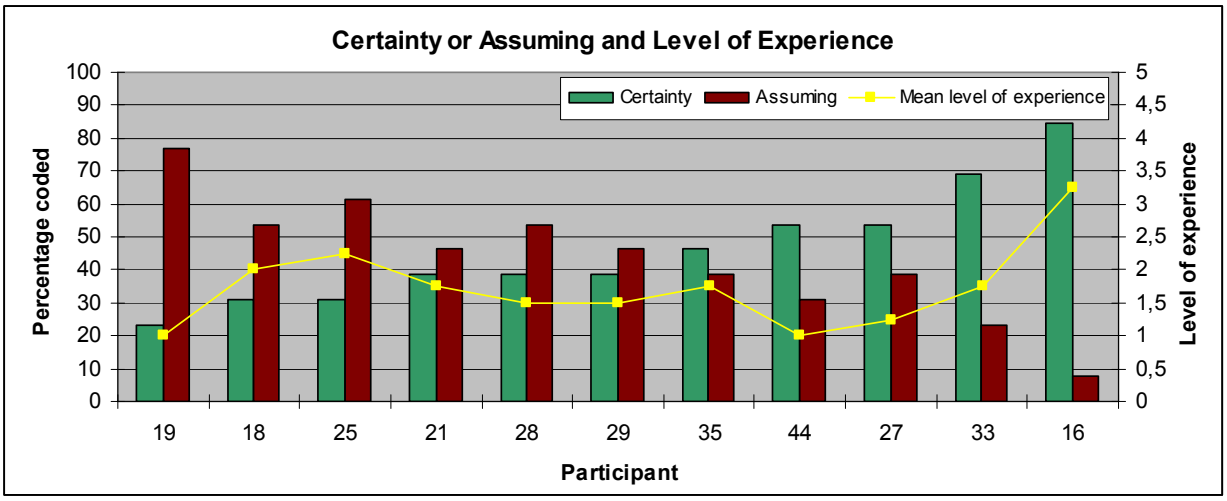


Diagram 3: Correlation between *certainty*, *assuming* and level of experience

Combining the results of both diagram 2 and 3 in diagram 4, we see that a higher level of experience only on average relates to the amount of correct answers. Outliers for example, are participant 27 and 44, who rate their experience below average but together perform a little above average. On the other hand we have participants 16 and 25 who rate their abilities above average but do not perform accordingly.

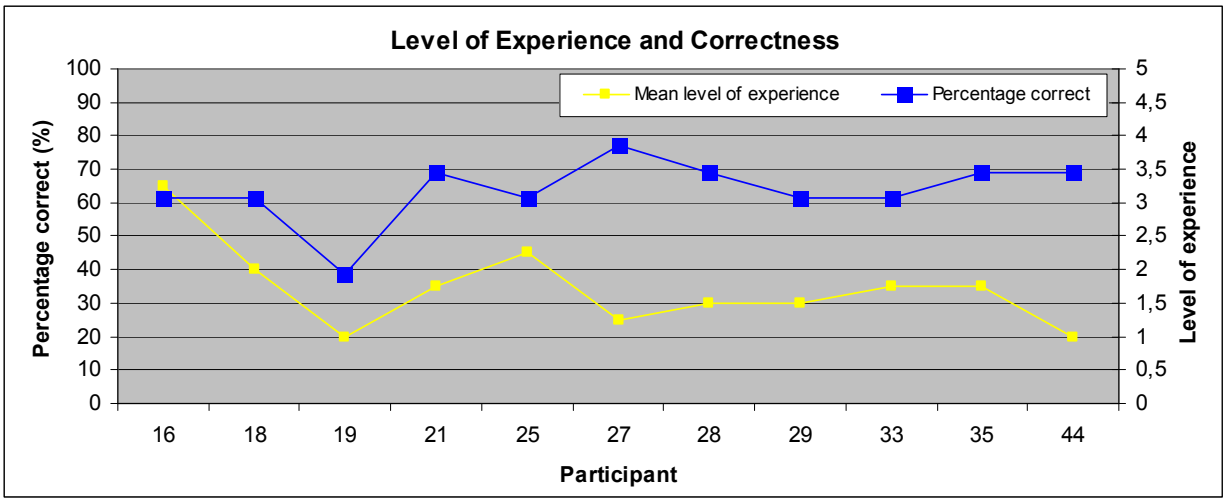


Diagram 4: Level of experience related to correctness

5.2.3.4. Related literature

Very little is known about how to interpret convictions verbalised by someone who answers a question. Research by Holmes [35] addresses the expression of doubt and certainty in English but she focuses on second language learners and the fact that for them it is especially difficult to master the linguistic devices that modify the information conveyed. Furthermore, her research aims to reveal the social aspects of expressing an attitude.

Nevertheless, a possible threat to validity upon the codes in this section is to be found in her study: "[L]earning to express and interpret modal meaning presents problems for second language learners" [35]. As most of the participants in our experiment were non-native English speakers, this is very well relevant in interpretation of the results.

Not so much related in this research, but very important in practice is the effect of the discussed behaviours on GSD. The desired behaviour of a developer is never to assume anything but to ask for clarification when something appears to him equivocal. Unfortunately, the distance in GSD negatively affects communication [14]: separate developing locations, dissimilar cultural backgrounds, a lack of common ground and deficient management all inhibit effective communication and lead to decisions differing strongly when developers

encounter ambiguity or incompleteness in the Software Architecture Documentation. Developers simply are less inclined to communicate with the designer as they do not know who it is or where he is situated [32, 21].

Sending out liaisons to establish common ground and personal relationships have been showed to increase productivity: knowing the context and knowledge of other teams increases effectiveness [40, 49] and establishes trust among teams [36]. Due to hindered communication, in GSD it is particularly important that every development team knows what to expect from related components produced by other teams. It are the *undetected* inconsistencies that cause the most problems [48]. Therefore, informal communication should be encouraged [7, 50] and documentation should be clearer and more explicit in conveying designs and decisions.

5.2.3.5. Conclusion, discussion and practical implications

First, relation between *certainty* and correctness was found, but the correlation between correctness and *assuming* was not. The fact that this last hypothesis was not corroborated by experiment could indicate that utterances used to convey uncertainty are a sign of a participant who guesses based on scarce, but apparently sufficient information to give the correct answer.

Second, some developers display a propensity towards one of both ends: they are either overly certain or overly assuming. This exemplifies unwanted behaviour: more assuming might indicate that the developer is less knowledgeable; more certain might indicate that the developer has an overrated self-image regarding development. A decreased correspondence between their abilities and their own *perceived* abilities could cause problems as it might increase the amount of unnoticed errors or mistakes. It then is only at integration of multiple pieces of code that it comes to light that the system does not behave like expected.

Third, it seems that there exists a relation between self-rated experience and skill, and confidence or attitude (*certainty* versus *assuming*): higher ratings of experience and modeling-skill appear to coincide with greater amounts of answers coded *certainty*. The other way around is corroborated by this research as well, however with less evidence. However, the outliers are more interesting as they provide us the exceptions to this rule.

Fourth, there is a general relation between mean level of experience and correctness of answers. Interestingly, part of the relation between self-ratings and correct answers was also found on quantitative basis in an analysis on a larger set of data from the same experiment, done by Heijstek *et al.* [30]. However, as mentioned earlier, it are the outliers that supply the more interesting information: here as well as with the tendency towards being certain or

uncertain an increased difference between self rated experience and correctness could be a cause for problems.

Different implicit assumptions about the communication between or responsibilities of components result in systems behaving erratic, and with problems of which the source could only be found with the investment of great effort. That is obviously not a wanted outcome. Being certain is good, be it on the right grounds only. Important is that the developer is able to notice the situations in which he himself is uncertain. This should probe him to ask the designer for disambiguation or explication of the unclear properties of the software architecture.

5.2.3.6. Future research

Future research should address the tendency of developers towards being certain or uncertain, whether or not this certitude is justified and what the effects are of such conviction on the decisions made during development. As proposed by Holmes [35], such attitudes could be projected onto a scale ranging to attain the *degree* of certainty expressed. This quantitative measure could subsequently be related to the *effect size* of, for example, defect density.

Other research could assess the self-awareness of developers about their attitude. We hypothesise that increased self-awareness results in better software, as the developer then knows when to ask for elucidation. Results could be used to construct awareness training for developers to increase efficiency.

5.2.4. Common Sense

5.2.4.1. Introduction

The documentation of a system should be explicit about every important or major property, especially when it comes to responsibilities of components and the properties of communication between them [47: 43]. Unfortunately, this is by far not always the case, and it is in these cases that developers have to choose how they react on such imperfection. Previous research shows that developers solve an inconsistency preferably by communicating with the creator of the documents or with other colleagues, but also that physical separation of a mere 50 meters results in the end of all regular communication [47, 33].

Our research takes this investigation one step further: not based on a survey but on empirical experiment, we try to assess what developers choose to do in such cases: do they acknowledge that they did not find the answer (implying that they would ask the designer for more information) or do they assume properties of the absent information?

Several kinds of information could be distilled from answers coded as *common sense*. This analysis looks at developers' behaviour: their reaction on ambiguous or absent information, and the types of utterances used forming an answer in order to infer their certainty or knowledge bases used.

5.2.4.2. Examples

In our study, typical behaviour for exceptionally unconfident participants was to use pre-existent knowledge about the domain of the system, or in short, use their common sense. This fragment of the answer of participant 19 on question Beta 3 constitutes a typical example:

> "*Yes, but what kind of message? Or, is it just general request? Ok. Well, yes, it shouldn't be ignored. Because the external request interface with the booking system which booking system send messages when a customer tries to book something online. He puts all his data, which are .. they're send to the server. So, sending messages. So, that's a request .. request messages. So it shouldn't be ignored.*"

From this short fragment is becomes clear that he does not know what to look for in the presented documentation. The importance of the messages he mentioned is rated crucial, but this assessment is based on nothing but common sense. The appraisal of importance is not bad in itself – for example, developers sometimes implicitly use the Level of Detail in a UML

diagram to successfully assess the importance of a component [47: 32] – but the source of information on which it is based is. The basis of this answer lies not in the documentation but rather in a personal perception of the domain of the system. Hence, the answer of participant 19 on this question was incorrect.

Another example of incorrect reasoning is to be found with participant 21, answering question Beta 3 as well:

> "*There's no special thing about this one. Messages can be ignored is a general.. It's unclear about this one so I guess yes, they can be ignored.*"

The participant infers some property of a component based on common sense. He probably thinks that the assumption that a message can be ignored is the safest option. But what if we are not talking about a system that manages bookings for flights? Of course we could suppose that the developer would have made another choice, knowing that the correctness and speed of the system is then of higher importance, but we will never be sure the developer will assume the same as we expect him to.

Participant 16 does the same when answering the Example question:

> *"Well, I see at first you have your table Task Log and for me it's very obvious to store your history there."*

As a last example, take a look at participant 29's answer on question Beta 3:

> "*Well, I see an external system request but I think this is the other way around. And I think such request can be ignored, but it's just my conception of what an external system is. And since we don't have any control over it, it can always be ignored. That's what I would say.*"

5.2.4.3. Analysis

Deviating from the general structure of the sections, most of the analysis was directly put besides the examples above. Nevertheless, generalisations about the relation between the codes *common sense* and *assuming* are still made here.

Except for participant 16's answer on the Example question, every other *common sense* answer is coded *assuming* as well. This indicates that uncertainty and the act of completing the information needed to answer the question with pre-existent knowledge go hand in hand.

Also, one could try to relate the use of common sense to viewing habits: the amount of switching between the two media presented and the time spent while answering the question. Unfortunately, in this small data-set no patterns based on this behaviour could be found.

5.2.4.4. Related literature

5.2.4.4.1. The Cognitive Theory of Multimedia Learning

According to Mayer [44] information that has to be understood and remembered by a learner, has to be presented in a concise manner. The Limited Capacity and Active Processing assumptions – explained in further detail in Appendix A1 – show that irrelevant information distracts the learner from the essential information and that the presentation of too much information creates a cognitive overload. This problem is twofold:

1. The distraction of the learner from important material by useless information;
2. The cognitive overload created by presenting too much information.

The first problem seems to contradict the hypothesis to document as explicit as possible, but this is merely a paradox. In the case of incomplete SADs, developers have to assume certain properties of the system: a developer only has to assume something when necessary information could not be found or is simply absent. Mayer assesses this problem as well: empirical evidence only showed absence of a positive effect of concise presentations over elaborate ones when the concise description of a system "may have been so short that it excluded some essential material." [44: 104]. Besides, the increased amount of information in SAD does not turn down the *coherence* principle: research showed that simplified diagrams most strongly support information integration, but those simplified diagrams are stripped of representational complexity only[1] [13].

[1] Complexity of diagrams can be divided into two different types: their essential and their representational complexity. *Essential complexity* refers to the minimal amount of information it should contain in order to convey

Second, a cognitive overload could be introduced when SADs present too much information. This cognitive overload can be reduced, especially by applying the *spatial contiguity* principle – in which corresponding information is presented in close proximity. The presentation of information in one diagram could be structured using this principle, but when even *more* information needs to be conveyed, other principles like *segmenting, signaling* and *pre-training* should be applied to reduce the strain on one's cognitive resources.

Learning consists of 3 processes [44: 71]: selecting, organising and integrating. This active learning takes place at the cost of cognitive processing. Reducing the need for processing (i.e. reducing the need for extraneous processing by taking the *coherency* principle into account) leaves attentional resources untouched, so they can be spent on essential and motivational processes in order to construct an enhanced mental model that ameliorates learning.
Based on the cognitive theory of multimedia learning, one would expect that less switches result in a less extensive and inferior mental model, requiring common sense to come up with any answers at all: when a learner switches often between the two media, he segments the data into bite-sized chunks himself, in order to be organised and integrated. It is only after understanding has occurred that the learner moves on to a next segment. Although opposite results have been found by Heijstek *et al.* [30], we cannot judge this because this research does not investigate upon this.

5.2.4.4.2. The conciseness of documentation

An upper limit of about two pages per artefact appears to be a reasonable amount: the majority of the developers simply do not want to read more than that [1]. So it should be noted that explicitness in documentation has its limits in use: within many organisations there exists a common ground, consisting of a characteristic corporate culture, shared knowledge and shared development methods [49]. Documentation about software systems probably may leave out elements commonly available to all developers through common ground in order to come up with more concise documentation. This does not conflict with the *coherence*-principle, because documents containing information that is part of the common ground, contain extraneous information, thus violating this principle. The

---

the required knowledge; *Representational complexity* refers to its fidelity and is made up of extraneous material that is irrelevant to the message [13].

combination of a well established common ground, easily accessible knowledge and concise documentation appears to fit (Global) Software Development best.

5.2.4.5. Conclusion, discussion and practical implications

5.2.4.5.1. Discussion

What we saw while examining this code was that of the 5 participants, 4 had their answer on question Beta 3 coded as using *common sense*. Probably this has something to do with the type of question or the domain of the system presented. Because this question is regarded difficult – its indeterminable nature and the implementation of a false friend[1] contribute to this – and the availability of expectations about how such a system should behave, make this question susceptible to implementation of knowledge regarded as *common sense*.
Second, the fact that the codes *assuming* and *common sense* coincide appears to be no accident: it could safely be assumed that when knowing less, one assumes more and while assuming, one integrates common sense and pre-existent knowledge into their answer.

5.2.4.5.2. Conclusion and implications

What could be read in section 5.2.3.4 also readily applies to this code: developers who make assumptions because some property that is unclear or ambiguous, express unwanted behaviour: different conceptions about what a component should do, ultimately clash at integration stage. In addition, when a design specification is outdated and a developer unknowingly does choose to use it, the same problems occur as well. In order to overcome this, Cataldo *et al.* advocate to promote lateral communication, identify dependencies amongst components and development teams and close the documentation-source code gap [15].

5.2.4.6. Future research

It might be interesting to investigate the relation between *assuming* and *common sense*, especially concerning the direction of this relation. Does an assuming nature of the developer implies increased use of common sense or is it the other way around? In the present research, *common sense* implies *assuming* in most cases but that does not show us its cause. Does a

[1] A false friend is a property that could be expected mislead someone. For example: there exit a component A with property a and a component B. The question asks about a and B. The participant then may be falsely inclined to ascribe a to B instead of the actual A.

familiar system domain automatically implies increased use of common sense and if so, why? Is it because the designer left out details he (actively or passively) expected to be known with the developer or is it because the developer is inclined to use knowledge that is most readily available to him – i.e. in his memory rather than in the documentation – an effect known as the *availability bias* [60]. Future research should be performed in order to assess these relational directions.

Still other research should address the correctness and completeness of the mental model. The contents of it are usually a mixture of perceived and retrieved information, but the proportion between them should be correct. We think that increased application of common sense implies a mental model with too much information from the LTM.

5.2.6. Single Medium Based Answer (SMBA)

5.2.6.1. Introduction

As with the previous codes, *SMBA* was chosen for its abundance. Furthermore, it appears to be a reliable code because matching cases are easily discernable. Besides the analytical advantages it is noteworthy to know whether 1 medium appears a subject to suffice in giving an answer.

5.2.6.1.1. Definition

Answers coded as *SMBA* have to conform to the following rules:

1. The participant is willing to base his answer on only one of both media when he is unable to find support for the answer in the other medium;
2. The participant only looks into one of the two media and comes up with an answer.
3. The participant looks into both media but only peeks shortly (presumably too short to see anything valuable) into one of the two, before uttering an answer.

Because this code relies on the transcribed switches in viewed media, those switches will be included in the following examples. [D] indicates that the participant is looking towards the diagram, [T] means that the participant looks into the text.

5.2.6.2. Examples

First an example of participant 16 answering question Gamma 1:

> "*[T] Hmhm.. I cannot find any details [D] on the diagram [T] [D] on the Back Office System component, of the responsibilities. [T] Therefore I'm now looking at the [text]. (…) [D] [T] The responsibility is that it contains all the data that is relevant to the Mid- and Front Office [D] [T] systems. That's my answer.*"

This participant did look into both media, but found out that the answer was not to be found in the diagram. This answer is an *SMBA* based on the first principle described in section 5.2.6.1.1.

Another example is of participant 29 on question Beta 2:

> *"[D] I see the Frontend at the left, and the only authentication I see is in a lower right corner. And then I see the component "User database" and I don't see it anywhere else in*

the big picture. So I would say no it cannot directly authenticate the user (…). So I don't think it's possible to directly authenticate the user."

Clearly, participant 29 did not even bother to look into the textual part of the SAD. This is an example of the second principle described in section 5.2.6.1.1.

5.2.6.3. Analysis

Because of the first property of described in section 5.2.6.1.1, the code called *cross-media check* (*CMC*) is not the opposite: according to that property, the participant could still look into both media, but come up with an answer based on only one medium[1]. With *SMBA*, the participant's incentive to look into the other medium – if he does so – is not to verify or falsify; answers coded as *CMC* are motivated by such goals.

Of all 92 answers coded *SMBA*, 39 given answers were incorrect, so those results contradict what one would expect: if someone only uses a single medium, inconsistencies in the documentation would presumably pass unnoticed, leading to unpredictable behaviour by the developer and hence to a system behaving erratic or not functioning at all. Despite this, most answers coded as *SMBA* are correct. This can be seen in diagram 5: only participants 19 and 28 had a larger amount of incorrect answers coded as *SMBA*.

[1] Thoroughness in general implies that the participant checks his/her initial answer in the other medium. This may give the participant:

1. extra certainty about the correctness of his/her answer;
2. give contradictions;
3. provide insight in the fact that the answer is only stated in one of the two media;
4. Relations between both media. This might be done in order to find extra information or to come up with relations not thought about yet.

This code is applied regardless of whether the cross-media check was successful or not. This code excludes the following case:

1. The participant does not find the answer to the question in the medium it looked in first and therefore looks into the other medium to find the answer. It should be noted that this and point 3 are both coded *SMBA*.

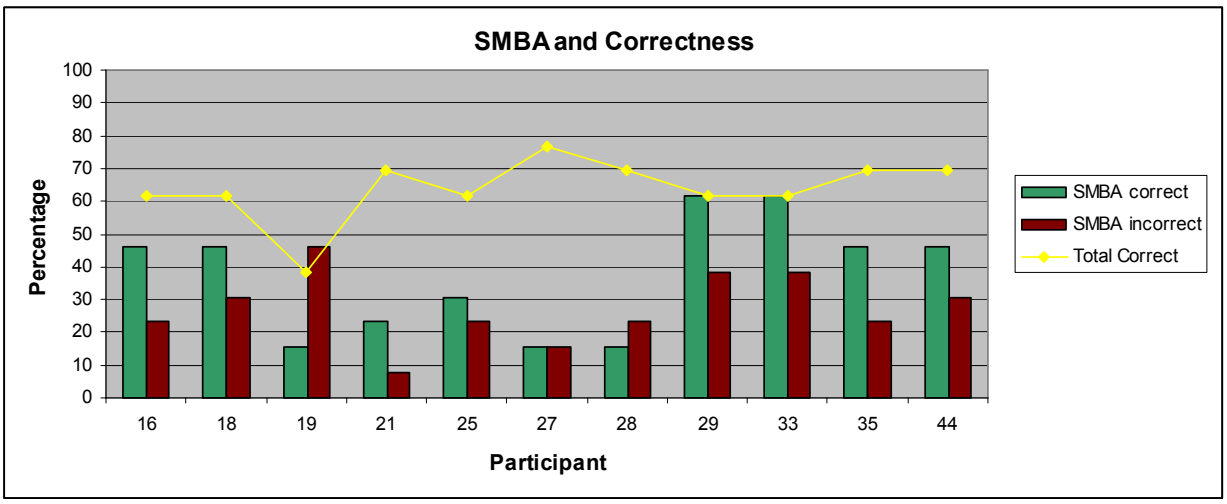


Diagram 5: *SMBA* and correctness

What we see in diagram 6 is that participants have a tendency to give answers coded *SMBA* or not: 6 participants gave 9 or more answers coded as *SMBA* covering a large majority of 64 out of 92 answers coded this way; the remaining participants stay around 5 answers coded as *SMBA*. Of those 6 participants, 24 of the 64 answers are incorrect, whereas 14 out of 28 answers of the remaining group are incorrect. Based on this small amount of evidence however, it seems that there is no real difference between people who tend to give multiple answers based on only one medium versus the other groups.

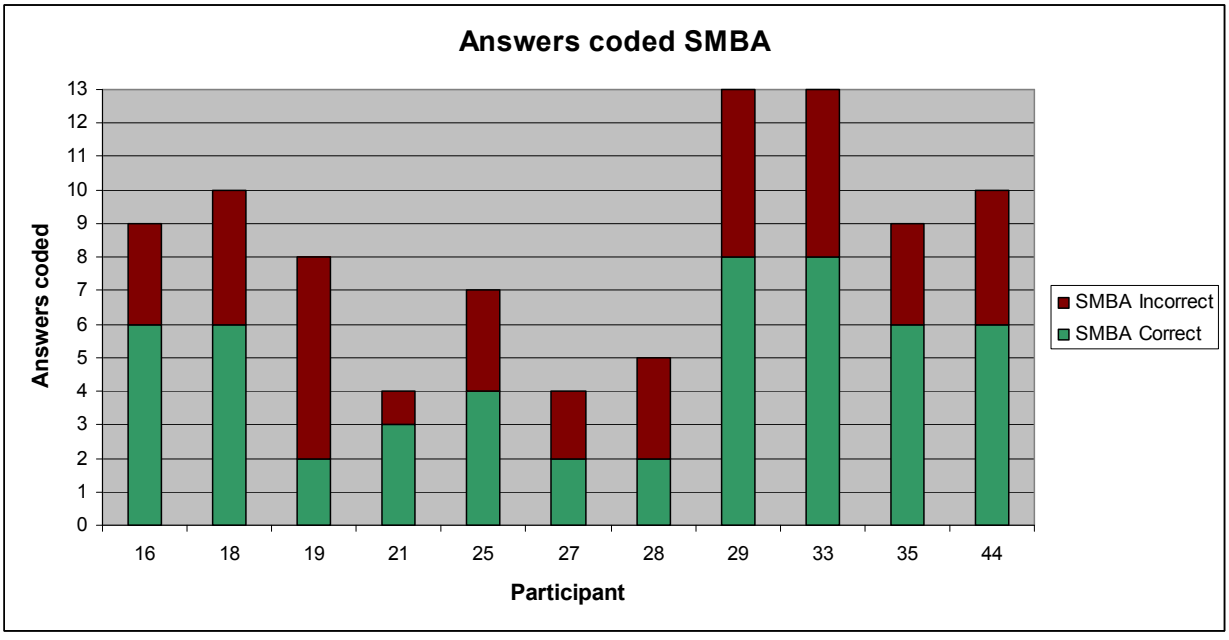


Diagram 6: Percentage coded *SMBA* per participant

The left two columns in diagram 7 display only a slight decrease in amount of correct answers coded *SMBA* when compared to the total average. But again, it should be remarked that answers coded as *SMBA* do not imply that the participant only looked into one medium. Rather, it shows that the participant is willing to base his answer on only one of the two media. What is interesting as well, the middle two columns show that participants who tended to give less than 9 (out of 13) answers *SMBA* were incorrect more often; participants with 9 or more answers coded *SMBA* reside at just the same level as the overall average. However, when leaving the restriction of answers coded *SMBA* behind however, the difference disappears and each participant – whether having less, or more than 9 answers coded *SMBA* – performs roughly the same.

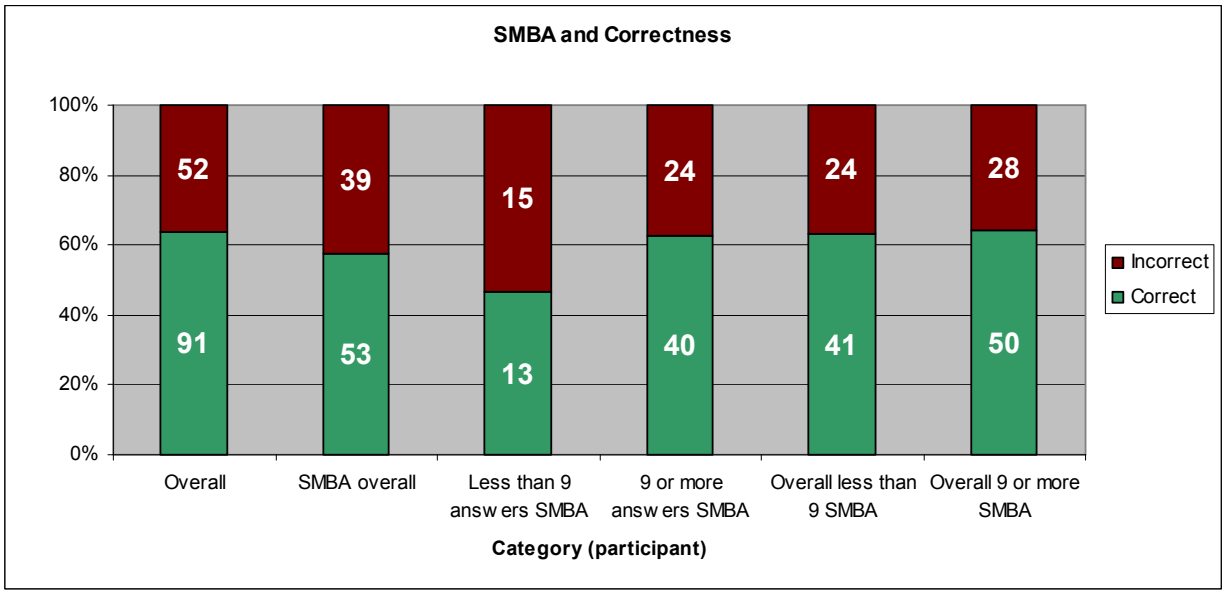


Diagram 7: *SMBA* and correctness per participant category. The middle two categories consist of the answers coded *SMBA* of respectively participants who gave less than 9 answers *SMBA* and more than 9 answers *SMBA*. The rightmost 2 columns show the same, but than for *all* answers of the participants divided into the same 2 categories. The numbers represent the actual amount.

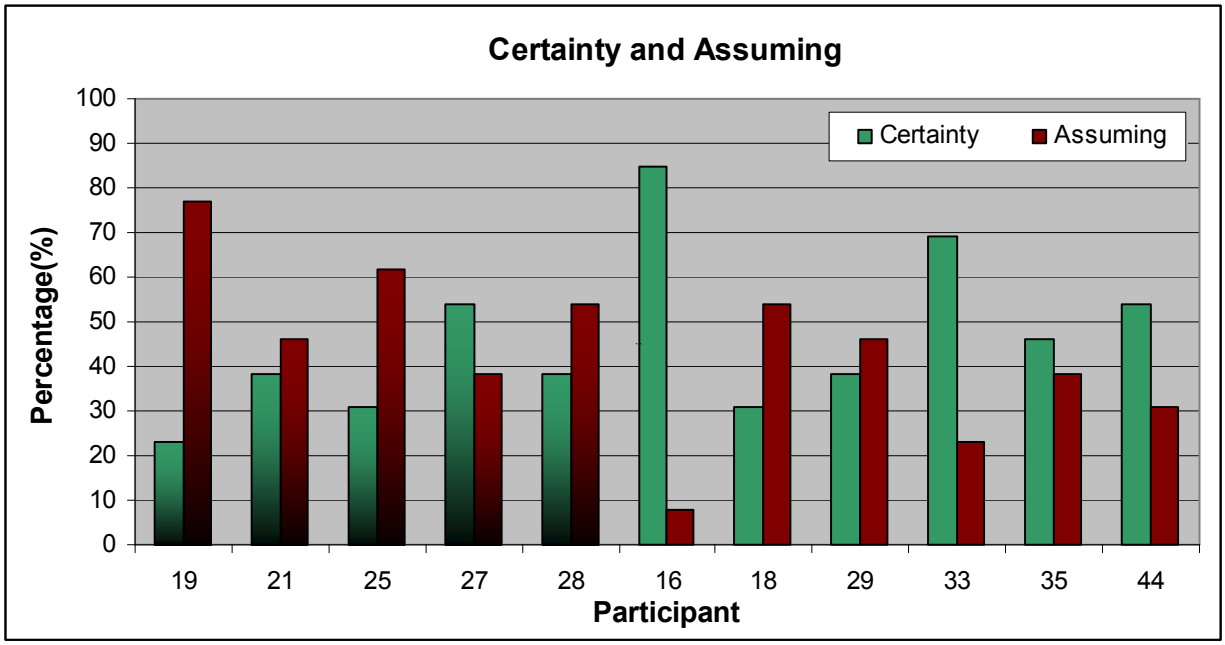


Diagram 8: *Certainty* and *assuming*. The left 5 columns are constituted (displayed using a gradient) by the participants in the 'less than 9 answers *SMBA*' group; the right 6 in the category of '9 or more answers *SMBA*'.

As follows from diagram 8, participants answering less than 9 answers using *SMBA*, are more often assuming than the participants in the other group. A possible explanation for this decrease in correctness could be that the participants were less certain about their answer. Hence, they construct or devise their answer using both media rather than just verbalising presented information. That they chose to *construct* answers could be due to the absence of required data, – but then every participant should show the same pattern – their inability to extract the requested information or that they excessively wanted to ensure themselves.

Results demonstrate as well that the questions Gamma 1 through Gamma 3 are all coded as *SMBA* at least 1 time more often than the total average. The participants here appear satisfied easier or are find it less problematic to use evidence found in only 1 document for a decisive answer. This could probably be ascribed to the architecture under question, but that is just a tentative guess. These questions do not contain any recognised differences compared to the other questions, so that leaves the architecture as the instigator of the difference in codes. It is only question Gamma 3 that is unanswerable and has a false friend implemented as well.
Furthermore, questions Alpha 3, Delta 1 and Delta 2 are coded *SMBA* 1 time less than average. When taking a closer look at those questions, they share a common denominator: they are all of topological nature. Apparently, people are more difficult to satisfy with such questions, or, answers for the mentioned questions are available only by combining both documents.

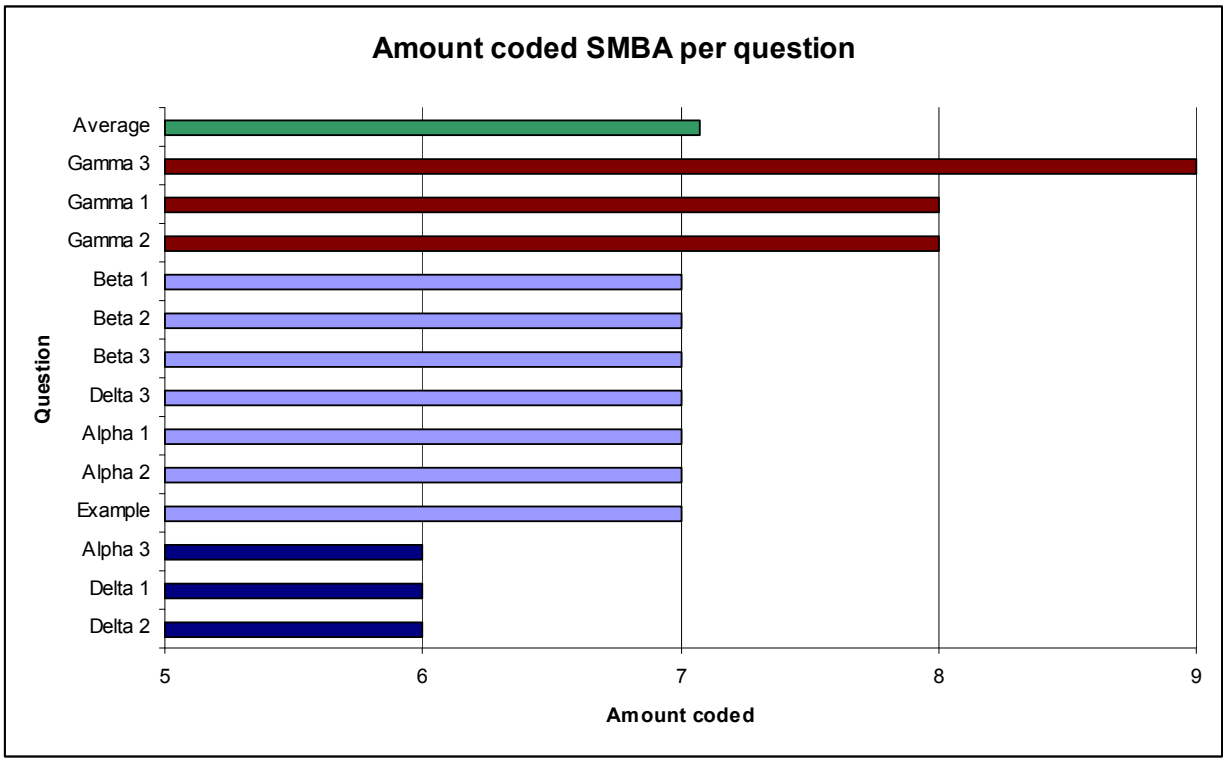


Diagram 9: Amount of *SMBA* per question

Remarkable is the difference between the categories 'less than average' and 'greater than average': both deviate from the total average as presented in diagram 10. Answers given in the first category are correct more often; answers in the second category less than average. From these results it can be deduced that there exists an inverted relation between the amount of answers coded *SMBA* and the amount of correct answers: questions that make a participant more susceptible to answer it using only one medium, increases the amount of errors. Still, different causes for the existence of this relation could be appointed. The present research however cannot provide conclusive thoughts regarding this.

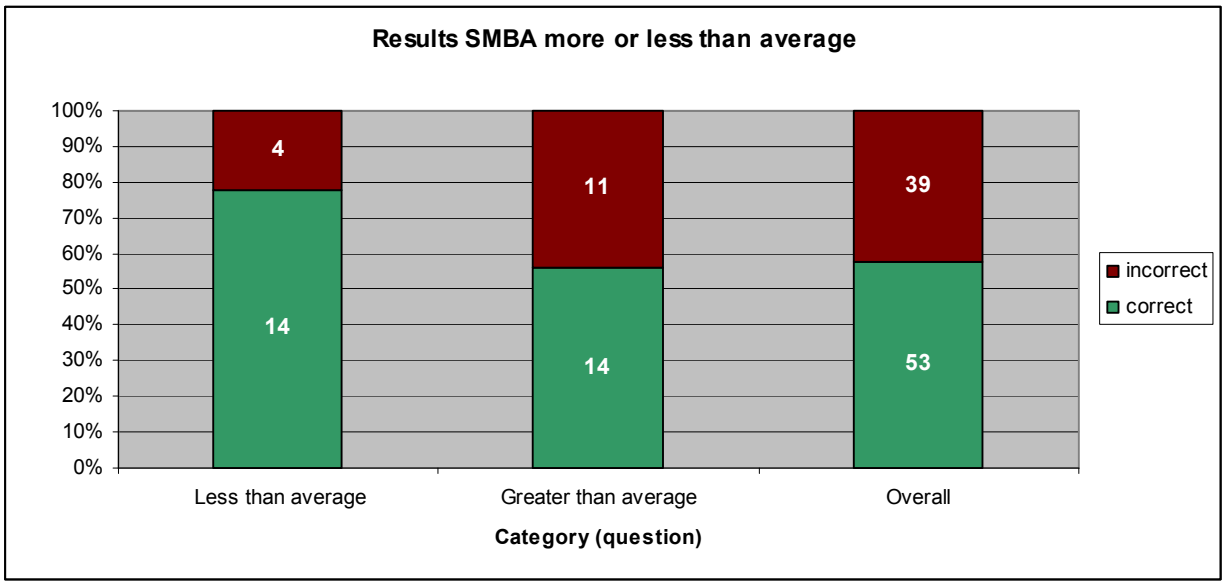


Diagram 10: Results of questions coded *SMBA*. The category 'less than average' are constituted by questions A3, D1 & D2; 'greater than average' are questions G1, G2 & G3. 'Overall' consists of only answers coded *SMBA* and corresponds to the second column in diagram 6. The numbers represent the actual amounts.

5.2.6.4. Related literature

In a study of Nugroho, it is tried to relate the Level of Detail of UML diagrams to problems at implementation. They found that a lower level of completeness in UML diagrams leads to deviations and hence problems in implementation [47: 46]. The current research however, has been performed with a wider scope: mainly looking at completeness and correspondence between multiple media-types involved in conveying software architectures through documentation. Therefore, the results of Nugroho and the current research complement each other, as textual descriptions are still needed to understand the documentation.

The conclusion of research by Kalyuga *et al.* is that experts learn better from information not integrated [39]. *SMBAs* do not support nor refute this because this is a disparate character from the expertise reversal effect: if an experienced participant tends to answer a question based on only one medium, it might even be helpful to include more elaborate information into the observed medium as it gives him access to all needed information using only the one medium. More information about the Expertise Reversal Effect can be found in section 9.1.

5.2.6.5. Conclusion, discussion and practical implications

A few conclusions can be drawn from the analysis above. First, answers coded as *SMBA* tended to be wrong slightly more often than average. This conclusion however, is limited due to the fact that the SADs did not contain any contradictions or incorrect information.

Second, participants who tended to give less than 9 (out of 13) answers *SMBA* surprisingly were incorrect more often than average. While the difference disappears when leaving the restriction of answers coded *SMBA* behind, it is highly interesting that participants with a tendency of answering *less* questions *SMBA* are incorrect *more* often. Further results subscribe to the viewpoint that these participants are uncertain: they use multiple media in forming (constructing, devising: i.e. not merely reading) an answer, hence no *SMBA* was applied.

Third, a few questions were noted to be coded as *SMBA* 1 time more or 1 time less than average. In addition, there is an inverted relation between the amount of answers coded *SMBA* and the amount of correct answers: it appears that questions that make a participant more susceptible to answer it using only one medium, increases their rate of error. Considered sources for these differences are variations in architectures and questions, but it lacks indisputable evidence, both here as well as in the previous paragraph.

Fourth, more than 60% of the answers were given *SMBA.* This SAD usage has some major implications on the construction of such documentation: a provocative question remains: is it profitable to use multiple media?

5.2.6.6. Future research

What was neither examined nor found, is how contradicting or incomplete SADs affect the quality of the resulting software. It might be interesting to research the effect of inconsistent architectural documentation in software development more deeply. A hypothesis – tentatively founded in the present research – is that there exists a disadvantage basing an answer on only one medium: incompletenesses or inconsistencies in or between the two media will pass unnoticed by the developer if he does not check his answer or is not able to check it. In some cases, this will lead towards inconsistencies in the developing system [47], but it is further research that should come up with a decisive answer.

Furthermore, the second conclusion in section 5.2.6.5 could be examined to a further extend: why is it that participants who answered less questions *SMBA* were less certain?

Also, the last observation needs further research: why choose people to use only one medium? Is integration between multiple sources too burdensome or are it personal preferences?

[2] As a part of the Cognitive Theory of Multimedia Learning. See Appendix A1, section 3 for more information.

5.3. Conclusions

The observation of participants while trying to solve the problems proposed, yielded lots of information, but even more ideas for future research. In depth and conclusions and accompanying justifications are drawn with each code, so only an overview of them is given here:

1. There is a positive relation between *certainty* and correctness but between assuming and incorrectness there is not;
2. Some participants leaned towards a certain or assuming attitude;
3. There exists a positive relation between self rated experience and modeling skill, and confidence or attitude. In addition, a positive relation between self-rated experience and modeling-skill, and correctness also holds;
4. Unanswerable questions with architectures with implemented false friends increase the amount of *common sense* applied;
5. Questions answered using *common sense* are in all but 1 case coded *assuming* as well;
6. Answers coded *SMBA* are slightly more often wrong than average;
7. Participants who tended to give less than 9 answers *SMBA* were incorrect more often than average;
8. Questions that increase a participant's liability to answer it *SMBA* increase the amount of errors made as well;
9. Some participants were answered (almost) all questions based on a single medium.

However, it are the outliers that supply the more interesting information. For example regarding conclusion 2 and 3, an enlarged difference between one's perceived skill and their actual skill could be a cause for problems.

The implications of these conclusions are manifold: developers behave in typical ways and have different levels of knowledge and expertise. However, today architectural documentation is the same for all those different people, which results in unpredictable variation in decisions made. This study investigated upon the sources and effects of participants using different procedures while answering questions in order to reduce the effectiveness of a developer to several of his characteristics. Knowledge about these characteristics enables researchers and software architects to improve documentation in such a way that a wider range of developers can use it in a more successful manner.

Apart from consequences for documentation, it became apparent that an increase of awareness about attitudes one has regarding decisions during software development might increase the quality of the product as well. Knowing better when to ask for explication, uncertainties and ambiguities in documentation are spotted earlier and consequently solved. This results in quicker and cheaper fixes and hence more stable software. However, further research is required to assess whether or not this is an actual possibility. Probably this is realisable through a training programme to raise this awareness.

At this point, the attentive reader might have missed the theory or hypothesis that Grounded Theory should have come up with (see Appendix B, section 2.2.2). However, the goal of *this* experiment in particular never was to devise something like that: a shortage of time resulted in a relatively small amount of analysed subjects (still not a problem *ibid.* section 3.1) and a somewhat more shallow analysis. Together with the acknowledged difficulty of GT (*ibid.* section 8.2) it was unfeasible to aim that high. Nevertheless, the findings – which could be regarded as hypotheses as well – show directives for future experimentation or more in depth analysis.

6. Analysis 2: Active Processing

6.1. Introduction

Following the Active Processing/Learning Assumption described in [44: 71], another type of analysis with encoding based on processes was performed. The transcript of one participant was analysed using a prescribed encoding vocabulary: the three phases of the Active Processing assumption[2] – *selecting*, *organising* and *integrating* – were treated as codes and applied to the transcript of participant 21.

As could be seen in figure 7, there exist three phases in the cognitive processing of information. These phases allegedly take place in consecutive order, and it should therefore be possible to discern them in a transcript of the verbalisations induced using Think Aloud/Talk Aloud.

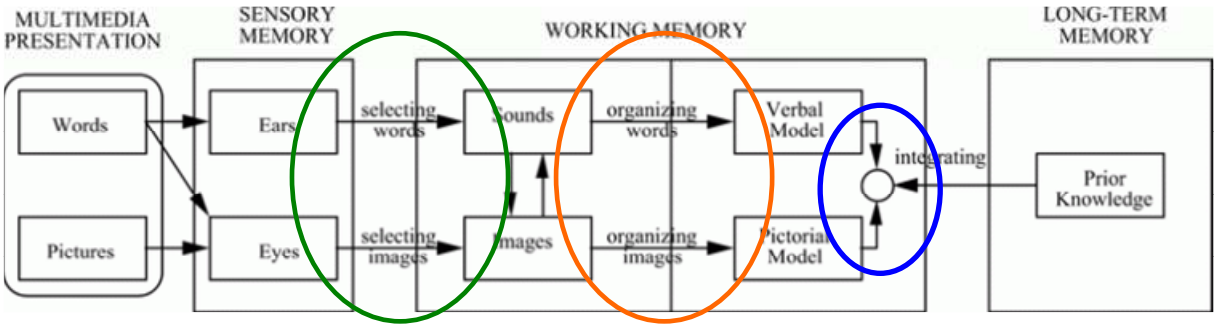


Figure 7: Phases of the Active Processing Assumption, adapted from [44: 61].

6.2. Theoretical Underpinnings

6.2.1. The participant chosen

Because this analysis is only additional to the main research, it was chosen to process only one participant. As could be read in Appendix B, section 3.1, excellent participants are typified by several properties [12: 231]. The one participant in this additional analysis was chosen based on these guidelines: participant 21 meets all the proposed requirements and was subsequently chosen. He was

- known with the phenomenon under investigation;
- willing to participate;
- articulate.

6.2.2. Encoding & Guidelines

6.2.2.1. Why coding?

As described in Appendix B, section 5, codes form the focus for thinking about the text and its interpretation [26: 40]. Because codes help to organise the relatively unstructured qualitative data, in this analysis the encoding approach is applied as well.

Following roughly the same approach as Ericsson and Simon [25] in their Verbal Protocol Analysis methodology, a prescribed ecoding vocabulary was used. The main difference however, between Ericsson and Simon's method and the approach applied here, lies in the fact that they performed a formal task analysis to derive an encoding vocabulary [25: 276]; we took existing labels of cognitive processes that take place, and applied those to the data available. Moreover, the application of equivalent terminology enables for incorporation of the results into the larger body of research, hence hopefully proving useful.

6.2.2.2. Guidelines for reliability

In order to reach as high levels of reliability as possible, several guidelines for the decision making processes are required. Guidelines like the ones described below can be equated to the process of saturation of codes in Grounded Theory (GT). However, while the codes emerging during GT are changing as saturation proceeds; the aim here is to define a set of rules by which the three codes can be defined or recognised in a transcript, especially the specific transcript in this analysis.

Several coding cycles were applied to the transcript in order to discover any deviations from the guidelines, and adjustments were made to the constituted rules when it was thought necessary, just like in the normal process of saturation [12: 117] – of course without shifting meaning: they still describe the same phenomena, but only become more articulated.

Below, the emerged guidelines for utterances that are encoded in one of the three categories are stated:

Selecting

- Cues for reading or searching ("So, I'm looking for…", "I'm reading…");
- Cues for direction ("I look at the diagram first…");
- Description of perceived material ("I see…").

Organising

- Description of one perceived medium (mostly in use to form the answer);
- Conclusions ("No, it's not related", "I found something again…").

Integrating

- Use of LTM (i.e. "I think it was..." or "I read somewhere that…" or "I just knew");
- Both media are combined or compared.

In general, utterances forming answers will not carry a code because the action "answering" is not covered by this model: after having finished all stages of active processing, the participant simply verbalises that final thought, which happens to be the answer. In some exceptions, the giving of an answer is incorporated in organisation or integration stages: it is in these cases that the answer will be coded accordingly. In general, these cases depict uncertainty (*assuming* in section 5.2.3.1.1) and transience of the answer given. Mere uncertainty will not get the answer coded accordingly. Answers where one or two media were used extensively while giving the answer will be coded as well, because these occurances depict the use of the presented material in organising or integrating an answer.

As portrayed in Figure , the transcript will be encoded following the three phases of active processing. They are depicted by three coloured ovals, using the same colour-code as the words below:

- **Selecting**
- **Organising**
- **Integrating**

However, because the distinction between the latter two processes might not always be clear, an additional code was introduced that combines the two processes into one group:

- **Organising & Integrating undiscernable**

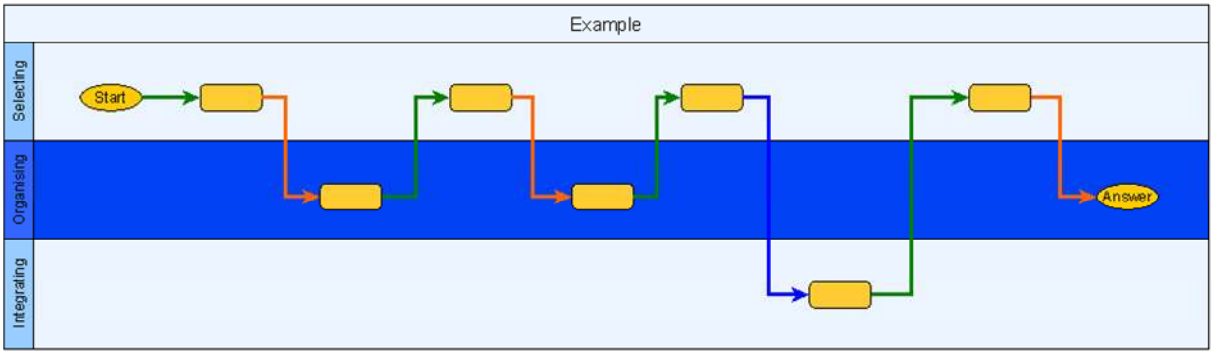
6.2.3. Hypothesis

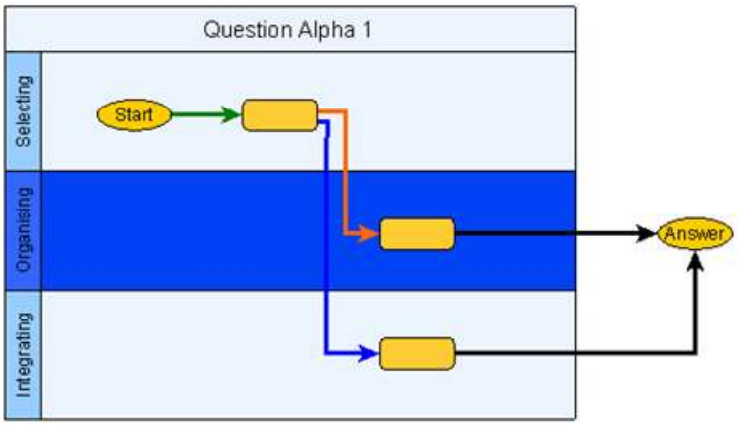
It would be expected that the participant first selects texts, then organises the obtained information and more or less simultaneously does the same for the diagram as well. After finishing these both processes, in order to compare and combine both sources and to come up with the (most) correct answer, integration would take place. Subsequently, the produced mental model is consulted to verbalise an answer.

6.3. Results[1]

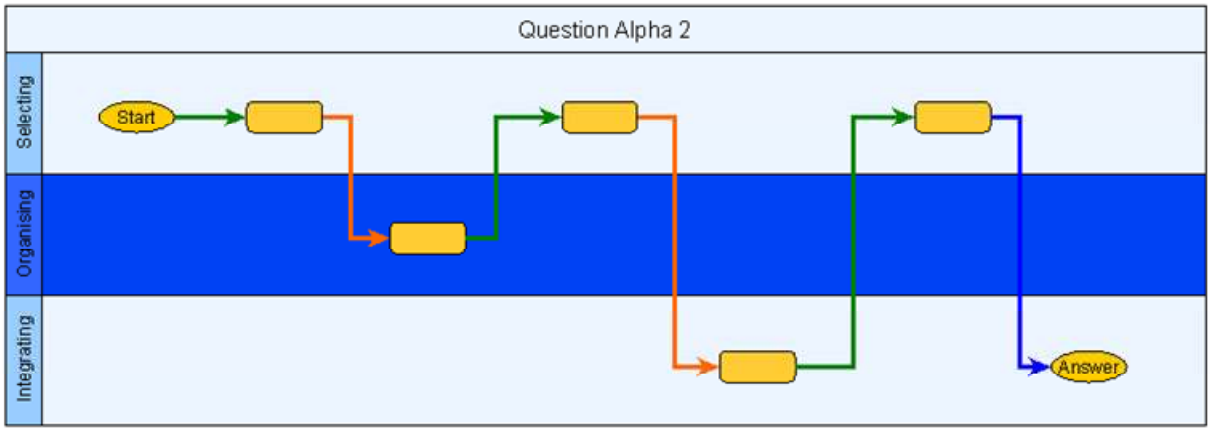
Not all answers were encoded, as some of them did not bear any useful verbalisations. This applies to the answers on question EQ32 and EQ43, which will be absent during further investigation. As could be seen in the transcript, the participant was not very outspoken and showed during these answers only a sequence of pauses and gaze-changes, hence leaving us uncertain about how to code the transcript of these answers.

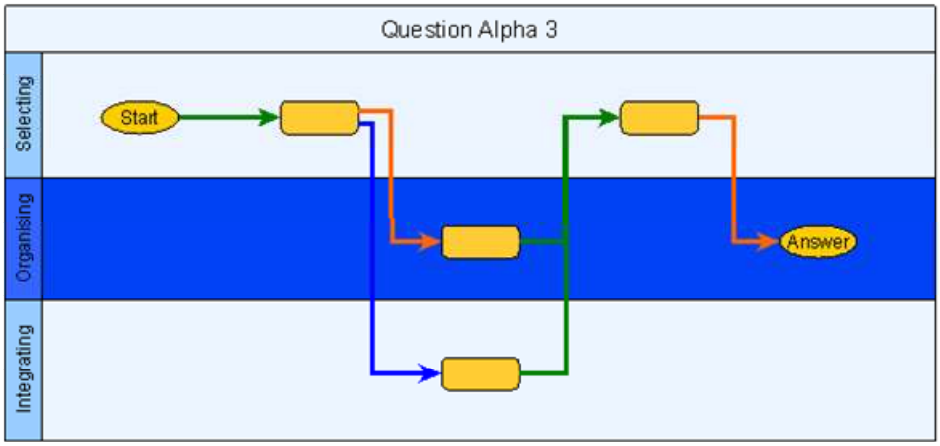
Below, all resulting state diagrams are showed. Each row depicts an activity of the Active Processing Assumption: *selecting*, *organising* or *integrating*. The arrows depict the same activities and the boxes the (unnamed) state in which the participant came out after that activity. Furthermore, the diagrams are to be read from left to right, where the ellipse at the left is the begin-state and the oval at the rightmost position the answer. Finally, there is no specific time-scale applied here.

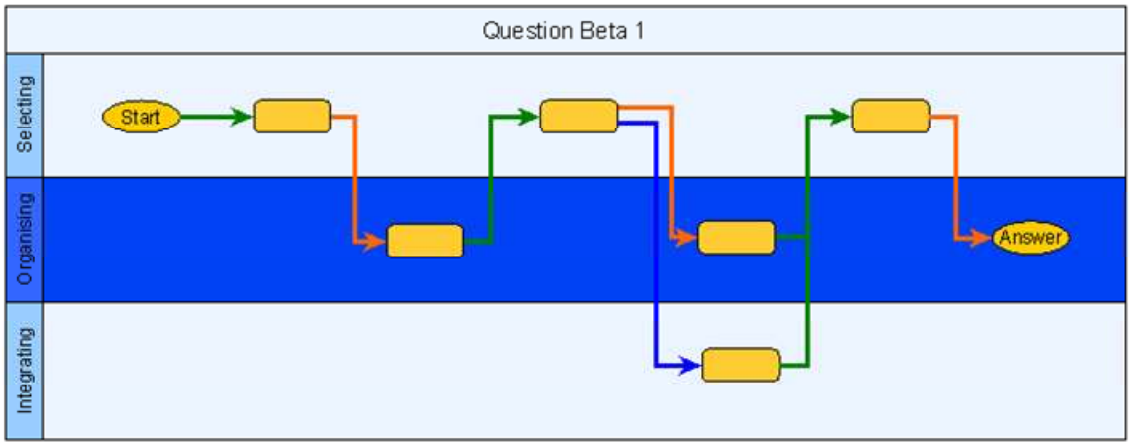


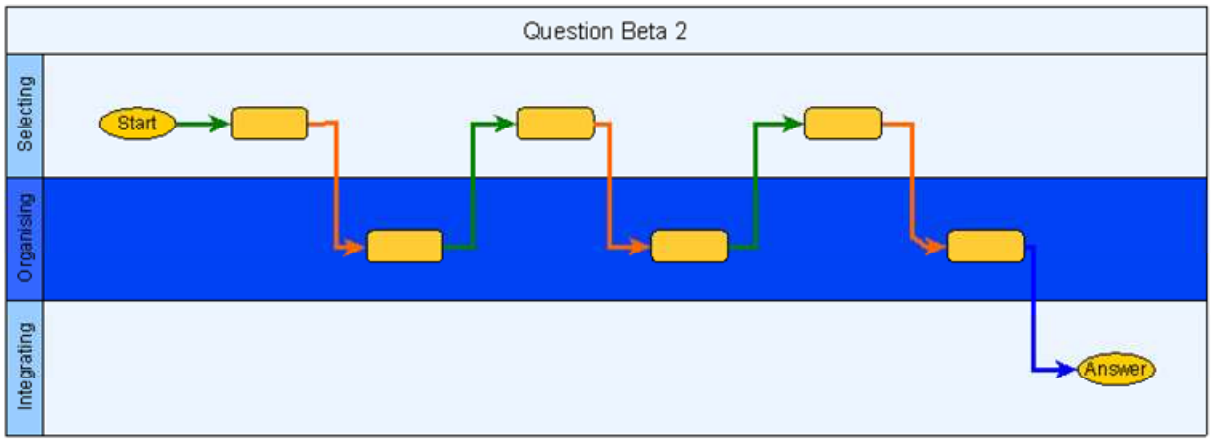


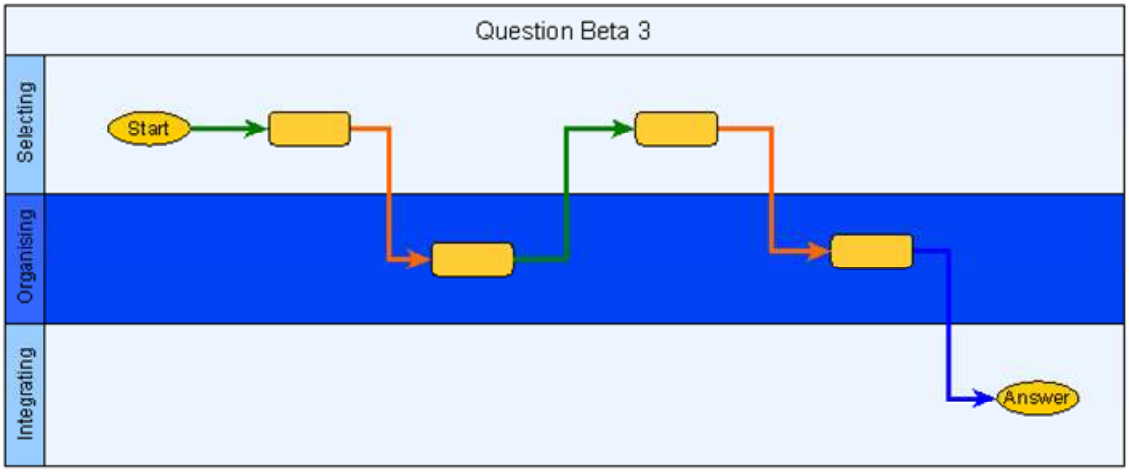
[1] The transcript can be found in Appendix C

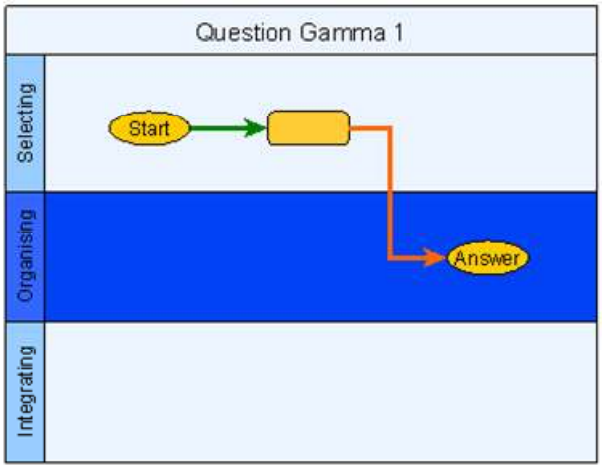

Question Alpha 2
Selecting
Organising
Integrating
Start
Answer

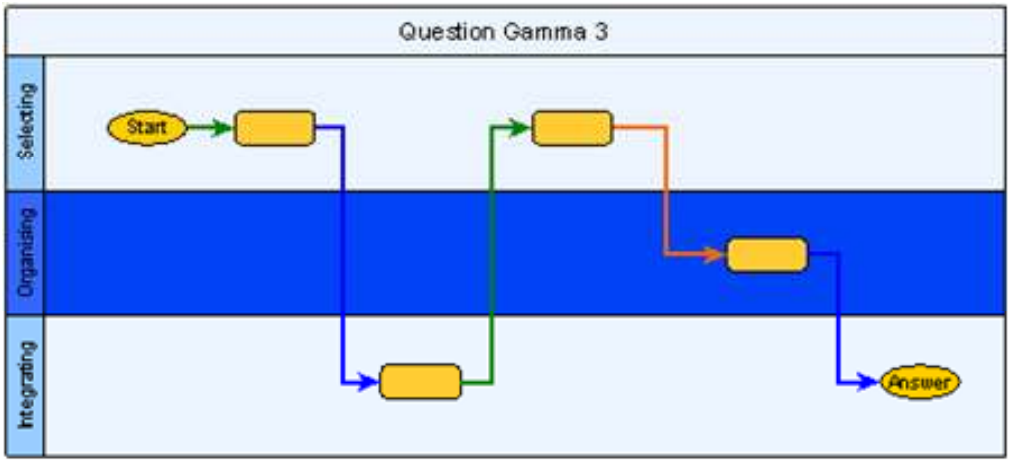

Question Alpha 3
Selecting
Organising
Integrating
Start
Answer

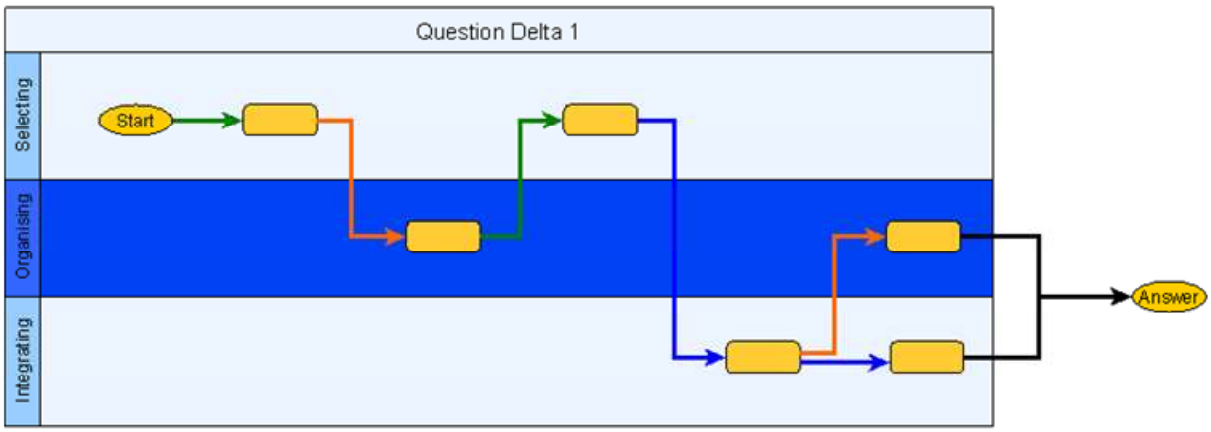

Question Beta 1
Selecting
Organising
Integrating
Start
Answer

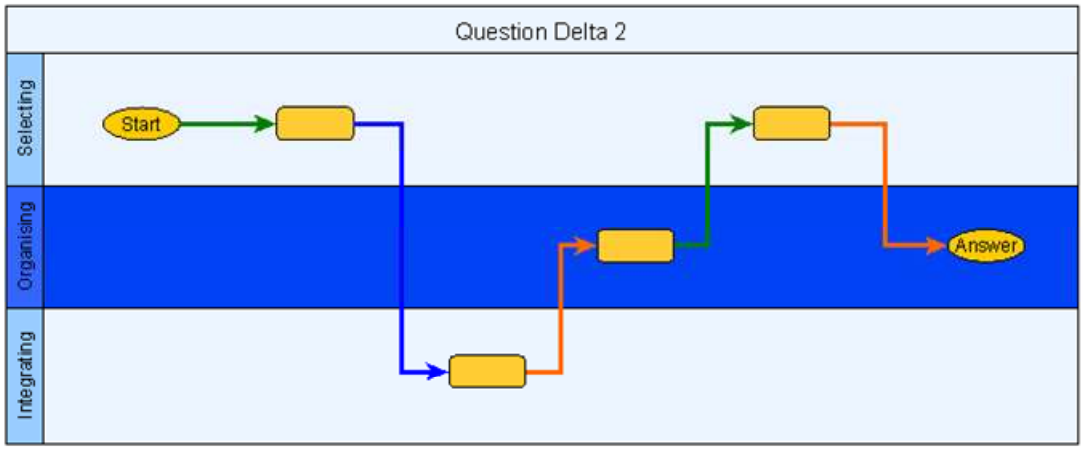

Question Beta 2
Selecting
Organising
Integrating
Start
Answer


Question Beta 3
Selecting
Organising
Integrating
Start
Answer


Question Gamma 1
Selecting
Organising
Integrating
Start
Answer


Question Gamma 3
Selecting
Organising
Integrating
Start
Answer


Question Delta 1
Selecting
Organising
Integrating
Start
Answer


Question Delta 2
Selecting
Organising
Integrating
Start
Answer

6.4. Conclusions

The previous diagrams show the activity pathways participant 21 took while answering the questions. This section will portray the conclusions that can be drawn from these results.

6.4.1. The initiation process

As could be expected, every process first engaged in was *selecting*. Simply put, the participant first needs to extract the information from the SAD to answer the question. However, in some cases this search probably was guided by information acquired with a previous question. A clear example of this is answer Gamma 3, where the participant mentions that he is searching for a specific value or property because he encountered it with the enquiry for an answer on preceding questions about the current architecture.

6.4.2. The final process

As described by the hypothesis in section 6.2.3, and therefore expected, not a single answer was given during the *selecting* process. However, 5 out of 11 times the answer was given in the *organising* state. This indicates that the LTM was not used in giving this answer, but also that there did not occur integration between the two media presented. This latter remarkable result might be ascribed to the fact that an answer was not always present in both media, hence forcing the participant to base his answer on a single medium. Another explanation hereof could be that the integration took place successfully and that the participant had an adequate mental model available for inquiry. Four out of these five instances endorse these stances by showing that the participant earlier tried to integrate information. Furthermore, all of these answers were correct, indicating that the participant's choice was a proper one.

In four cases, the last process was an *integration*-one. Two out of four answers meeting this property where incorrect. However, *every* participant examined in the previous more elaborate research[1] had these specific answers (Beta 3 and Gamma 3) incorrect, so without knowing whether or not these questions misled the participants in using the wrong strategy in answering them (as could be found by analysing the other participants in depth as well) or simply because these questions where indistinctly put, no conclusions can be drawn from this result. It could nevertheless be deduced that integration as a final process is undesirable. That late integration is annulling for correct answers could be due to the fact that this overdue

[1] Section 5

integration is one's last expedient: the participant is still gathering information to construct a sound answer.
The remaining two cases – answers on question Alpha 1 and Delta 1 – are too indistinct to be put into one of the three categories.

6.5. Limitations

Based on the guidelines in section 6.2.2.2, there might appear to be a clear distinction between organising and integration, but there is not. In some cases the transcript simply could not provide a decisive answer on whether the participant used 1 or both media: he could have looked into both media but still chose to use the information from only 1 medium (see *SMBA* in section 5.2.5).
Because the indistinctness between the different codes, the order of processes in which each answer was given, might not always be completely correct. This in turn could lead to unexpected outcomes, of which the fact that several answers were given in the *organising* stage rather than the expected *integrating* stage might be one. Furthermore, the conclusion drawn based on this the analysed information is equivocal as well: the conclusion that acquired information was not integrated *during* the presenting of an answer does not mean that integration did not take place. It might be the case that integration took place earlier and that the integrated mental model was already available in the working memory.

6.6. Future research

Based on the limitations expounded in section 6.5, adaptation of the straight on model of figure 7 for this specific type of experimentation and analysis seems indispensable. Future work should address the applicability of this model and aim to find better ways of inquiry (i.e. to improve the method of information acquisition), and to improve the correspondence and aptness of the Active Processing model with the data.
One suggestion involves an addition to the existing coding vocabulary. It could prove useful in discriminating between integrating WM models and retrieving previous knowledge from the LTM to add a code called *retrieval*. In discerning this process from *integration*, higher quality of data is obtained. The fitting in of new information – whether from LTM or the SAD – can then just be coded as *integration* and information fetched from LTM as *retrieval*.

The use of this vocabulary consisting of the processes in the Active Processing assumption, combined with research on general-purpose encoding schemes could be useful as template to determine the basic structure of a coding scheme [37]. Because one should avoid too much *ad hoc* changes to a theory – as it then looses its scientific value. Application of any arbitrary variation to a theory would render it unfalsifiable: it is then unable to rule out anything [54, 25: 264] – such general-purpose coding scheme should be devised prior to its application.

Another idea for future research involves the incorporation of a time scale; not merely states and transitions between them. The transcriptions were performed as an abstraction of the rich data, preserving only the relative sequence of processes. Measurement of the actual time spent during a specific process might then be an indication of invested effort. It should be noted however, that independent measurements of speech and read rates then are necessary to normalise the data of the experiment [25: 250].

Because much more information could be distilled from the material obtained using Think Aloud/Talk Aloud, this type of analysis should be done in further depth as has been performed here. The limited time resources forced us to focus this analysis on only one participant, but nevertheless it was tried to explicate the method used in detail so future investigation could be done using the described techniques as used here.

7. Observations & Conclusions

It is tempting to overestimate the impact of the results, but we need to absent ourselves from overgeneralisations. By positing the conclusions within our research, some interesting observations were made. More in depth justifications of these conclusions can be found in the previous sections of this thesis.

- **Conclusion 1.1:** *There is a positive relation between* certainty *and correctness but between incorrectness and* assuming *there is not;*
- **Conclusion 1.2:** *Participants show a tendency in attitude.*
- **Conclusion 1.3:** *There exists a slight positive relation between self rated experience, modeling skill and confidence or attitude. In addition, a positive relation between self-rated experience and modeling-skill, and correctness also holds;*
- **Conclusion 1.4:** *Unanswerable questions combined with architectures with implemented false friends increase the amount of* common sense *applied.*
- **Conclusion 1.5:** *Questions answered using* common sense *are in all but 1 case coded* assuming *as well;*
- **Conclusion 1.6:** *Answers coded* SMBA *are slightly more often wrong than average;*
- **Conclusion 1.7:** *Participants who gave less than 9 answers SMBA were incorrect more often than average;*
- **Conclusion 1.8:** *Questions that inclrease a participant's liability to answer it* SMBA *also increases the amount of errors made;*
- **Conclusion 1.9:** *Some participants were inclined to answer almost all questions based on a single medium.*
- **Conclusion 2.1:** *Answers given in the integration-stage are undesirable.*

8. Limitations & Threats to validity

The analysis of qualitative data is only as good as the way the data was collected [18]. Moreover, inherent to any type of qualitative research is the interpretation of data by the researcher – who is in his turn again influenced by his experience and background. By devising experiment & analysis protocols and following them meticulously, it was tried to ameliorate the data quality. In spite of this, some threats to validity or advisable modifications to our research could still be appointed:

- During the experiment, we used an undetermined amount of time before prompting the subjects to keep verbalising. This resulted in gaps in the recorded data, rendering some answers unanalysable. "A short amount of time" should have been more specific.
- Purposeful sampling should have been applied to find subjects that were most interesting to analysis 1. Instead, apart from the selection on properties of an excellent participant, we applied convenience sampling. This might have caused some interesting results to remain covered.
- Little is known about how to interpret conviction verbalised by someone who answers a question. Research by Holmes [35] addresses the expression of doubt and certainty in English. A possible threat to validity upon results could be found in her study: "[L]earning to express and interpret modal meaning presents problems for second language learners" [35]. As most of the participants in our experiment were non-native English speakers, this is very well relevant to interpret the results.
- Another threat to validity is attributed to the reliability of codings[2]. By constant comparison of the already encoded transcripts with the transcript currently being processed and the application of a fairly broad context of one answer as a segment, we did our best to neutralise any possible cause for bias. Therefore the recommended second encoder was not applied [37], and neither was multiple encodings based on different segment sizes [18].

On the other hand, the application of two different methods of analysis might increase validity – that is, if they do not contradict each other. *Triangulation* – the act of using different sources to verify results from different viewpoints – could then be applied to improve research validity

[2] Appendix B, section 6.4.5

[26]. Despite this, we did not construct any such direct link between the results of both analyses, mainly due to the limited commensurability of the codes. I borrowed the meaning of the term *commensurability* from Kuhn's work ([41]) in relation to paradigms: two subsequent major paradigms are incommensurable because their notions of elementary concepts are fundamentally different. With respect to our research, we applied two different "frameworks" (or vocabularies) applied on dissimilar levels of behaviour. Although the difference being not as fundamental as with paradigms, straightforward translation is not possible either.

Besides all actual limitations above, the appearance of an overwhelming amount of "counter examples" to the cases made might lead us to think this is a limitation as well. At this point, it is interesting to look again at one of the key characteristics of qualitative research: GT, along with other types of qualitative research, does not rely on notions of statistical representativeness to make claims about the generalisability and authenticity of the findings. Using particularly deviant cases in the main results ensured us to miss no important cases that might lead to question the outcomes[1].

[1] Appendix B, section 4.1

9. Implications & Future research

9.1. Direct & indirect implications

Proper documentation should decrease the amount of ambiguous understandings of a system. For example, false friends should be put more distinct to reduce their fake "friendliness". Where the purely experimental implications reach their boundaries, together with the performed literature research, these implications could be extended much further. Combining our research with literature on multimedia learning [44], some ideas come to mind that might help to improve its communication[1]:

- Redundancy might improve learning & understanding, but only when narration and text are presented separately. This could be applied through segmenting (dividing a SAD into smaller graspable units) or pre-training (provide a verbal description of the system prior to providing the SAD);
- Use signaling words, but sparingly;
- Spatial contiguity of related or highly coupled elements eases understanding;
- Integration of instructional guidance could be overdone[2]: Kalyuga *et al.* demonstrated that experts sometimes exhibit decreased performance with integrated guidance as it conflicts with their existing cognitive schemas. As a result, high-experienced learners obtain extra cognitive load by trying to cross-reference both representations or by trying to ignore the redundant information. The most important instructional implication of this effect is that instructional design should be tailored to the level of experience of intended learners [39].

Apart from the cognitive advantages that could be gained, increased formalisation of the graphical language enables for more extensive and computerised type checks to decrease the amount of inconsistencies [22]. Furthermore, the cost of creating, adjusting and using design documentation should be decreased: if documents are always up-to-date and easy to find, they could become a way of communication [42, 19].

[1] Appendix A1, section 5.3 & Appendix A2, section 3

[2] Appendix A1, section 5.2.2

9.2. Future research

Virtually every implication given above is in need of at least some further investigation. However, some ideas exceed even the propositions in the previous section. Besides, the new questions raised due to insights gained by the results of our experiment have to be assessed by further experimentation as well.

9.2.1. Exploring the experimental results

For example, future research should assess the direction of correlation between *common sense* and *assuming*: discrimination between the source and its consequence could prevent from treatment of symptoms.

Second, other research should be addressed to the tendency of developers towards being certain or uncertain, whether this certitude is justified and what the effects are of such conviction. Related to this, it might be interesting to somehow measure the awareness of their certitude. Results could help developers prevent from making inequitable decisions based on incomplete or ambiguous information. In addition, further research should shed light upon the feasability of a training programme for developers to raise awareness about their attitudes regarding decisions based on SADs: knowing better when to ask for explication, uncertainties and ambiguities in documentation are spotted earlier and consequently solved. Increased awareness therefore could decrease both cost and time to market.

Third, the effects of contradicting or incomplete SADs on the quality of the resulting software should be examined. Our hypothesis is that it is a major disadvantage basing an answer on only one medium: incompletenesses or inconsistencies in or between the multiple media will pass unnoticed by the developer if he does not check his answer or is not able to check it.

The additional analysis also requires some extra deepening. First, expansion of the used vocabulary with specific codes – to enable for higher resolutions in data and analysis – might be an interesting step. Second, an incorporated time scale might indicate the effort invested on each cognitive process.

9.2.2. Other suggested research

Apart from elaborating the results from our experiment, some ideas for connected research came to mind. A first example addresses consistency within SADs. This might be appointed using dynamic signaling: when a developer selects a component in the diagram, accompanying text should be selected as well. This eases his integration-stage and as a result facilitates consistency checks by the developer. In addition, while the designer should introduce the relations between documents explicitly, he is forced to check consistency as well. Apart from highlighting relations *across* documents, relations and dependencies *within* one document should be dynamically signalled as well: with the selection of one component, all related components should become highlighted to increase visibility of dependencies.
Assessment of the different levels of knowledge and expertise per developer should allow designers to fit a system's design to the developer's needs. Or, even better: tools should provide functionality to capture different levels of detail (LoD) to enable the developer to choose his preferred type of documentation and LoD.

An idea for an actual experimental setup could be to provide a participant with a SAD and instruct him to convert it into some sort of scaffolding code. While he is busy, he should be interrupted at various points in time or progress and asked about his current goals, sub goals and actions performed in order to obtain his current understanding of the described system. This task is more realistic than our experiment, so more reliable results could be expected. Besides, more detailed data could be obtained than the application of mere ToL did in our experiment. Several of the research directions proposed in section 9.2.1 could be assessed using this experiment.

10. References

[1] AGERFALK, P. J., AND FITZGERALD, B. Flexible and distributed software processes: Old petunias in new bowls? *Communications of the ACM 49*, 10 (October 2006), 27–34.

[2] ARMSTRONG, D. J. The quarks of object-oriented development. *Communications of the ACM 49*, 2 (February 2006), 123–128.

[3] BADDELEY, A. Working memory. *Life Sciences 321* (1998), 167–173.

[4] BADDELEY, A. D. Is working memory still working? *European Psychology 7*, 2nd (June 2002), 85–97.

[5] BADDELEY, A. D. *The Essential Handbook of Memory Disorders for Clinicians*. John Wiley and Sons, 2004.

[6] BADDELEY, A. D. Working memory. *Current Biology 20*, 4 (February 2010), 136–140.

[7] BATRA, D. Modified agile practices for outsourced software projects. *Communications of the ACM 52*, 9 (September 2009), 143–148.

[8] BIANCHI, A., CAIVANO, D., LANUBILE, F., RAGO, F., AND VISAGGIO, G. Distributed and colocated projects: a comparison. In *Proceedings of the seventh workshop on emperical studies of software maintenance* (November 2001), pp. 65–69.

[9] BIRD, C., NAGAPPAN, N., DEVANBU, P., GALL, H., AND MURPHY, B. Does distributed development affect software quality? an empirical case study of windows vista. *Communications of the ACM 52*, 8 (August 2009), 85–93.

[10] BOOCH, G. *Object Oriented Analysis and Design. With applications*, 2nd ed. Addison Wesley Longman, 1994.

[11] BROOKS, F. P. No silver bullet: Essence and accidents of software engineering. *IEEE Computer 20*, 4 (1987), 10–19.

[12] BRYANT, A., AND CHARMAZ, K., Eds. *The SAGE Handbook of Grounded Theory*. Sage Publications, 2007.

[13] BUTCHER, K. R. Learning from text with diagrams: Promoting mental model development and inference generation. *Journal of Educational Psychology 98*, 1 (2006), 182–197.

[14] CARMEL, E., AND AGARWAL, R. Tactical approaches for alleviating distance in global software development. *IEEE Software 18*, 2 (March/April 2001), 22–29.

[15] CATALDO, M., BASS, M., HERBSLEB, J. D., AND BAS, L. On coordination mechanisms in global software development. In *International Conference on Global Software Engineering* (2007), pp. 71–80.

[16] CATALDO, M., WAGSTROM, P. A., HERBSLEB, J. D., AND CARLEY, K. M. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Computer Supported Cooperative Work: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (November 2006), pp. 353–362.

[17] CHARMAZ, K. *Constructing Grounded Theory*. Sage Publications, 2006.

[18] CHI, M. T. H. Quantifying qualitative analyses of verbal data: A practical guide. *Journal of the Learning Sciences 6*, 3 (1997), 271–315.

[19] CHISAN, J., AND DAMIAN, D. E. Towards a model of awareness support of software development. In *The 3rd International Workshop on Global Software Engineering* (May 2004), pp. 28–33.

[20] CONCHÚIR, E., AGERFALK, P. J., OLSSON, H. H., AND FITZGERALD, B. Global software development: Where are the benefits? *Communications of the ACM 52*, 8 (August 2009), 127–131.

[21] CURTIS, B., KRASNER, H., AND ISCOE, N. A field study of the software design process for large systems. *Communications of the ACM 31*, 11 (November 1988), 1268–1287.

[22] DUKE, D. J., DUCE, D. A., AND HERMAN, I. Do you see what i mean? *IEEE Computer Graphics and Applications 25*, 3 (May/June 2005), 6–9.

[23] ERICKSON, J., AND SIAU, K. Theoretical and practical complexity of unified modeling language: Delphi study and metrics analyses. In *25th ICIS* (2004), pp. 183–194.

[24] ERICKSON, J., AND SIAU, K. Can uml be simplified? practitioner use of uml in separate domains. In *Proceedings of the Workshop in Exploring Modeling '07* (2007).

[25] ERICSSON, K. A., AND SIMON, H. A. *Protocol Analysis: Verbal reports as data*, 2nd ed. MIT Press, 1993.

[26] GIBBS, G. R. *Analyzing Qualitative Data*. Sage Publications, 2007.

[27] HADAR, I., AND LERON, U. How intuitive is object-oriented design? *Communications of the ACM 51*, 5 (May 2008), 41–46.

[28] HAYES, J. H. Do you like piña coladas? how improved communication can improve software quality. *IEEE Software 20*, 1 (2003), 90–92.

[29] HEALY, A. F., PROCTOR, R. W., WEINER, I. B., FREEDHEIM, D. K., AND SCHINKA, J. A., Eds. *Handbook of Psychology: Experimental psychology*. John Wiley and Sons, 2003.

[30] HEIJSTEK, W., KÜHNE, T., AND CHAUDRON, M. R. V. An experiment in media effectiveness in software architecture representation. 2010.

[31] HERBSLEB, J. D. Global software engineering: The future of socio-technical coordination. In *FOSE'07* (2007).

[32] HERBSLEB, J. D., AND GRINTER, R. E. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st International Conference on Software Engineering* (1999), pp. 85–95.

[33] HERBSLEB, J. D., AND MOCKUS, A. An empirical study of speed and communication in globally distributed software development. *IEEE transactions on Software Engineering 29*, 6 (June 2003), 481–494.

[34] HERBSLEB, J. D., AND MOITRA, D. Global software development. *IEEE Software 18*, 2 (March/April 2001), 16–20.

[35] HOLMES, J. Expressing doubt and certainty in english. *RELC Journal 13*, 9 (1982), 9–28.

[36] HOLSTRÖM, H., FITZGERALD, B., AGERFALK, P. J., AND CONCHÚIR, E. . Agile practices reduce distance in global software development. *Information Systems Management 23*, 3 (June 2006), 7–18.

[37] HUGHES, J., AND PARKES, S. Trends in the use of verbal protocol analysis in software engineering research. *Behaviour & Information Technology 22*, 2 (2003), 127–140.

[38] KALYUGA, S. Knowledge elaboration: A cognitive load perspective. *Learning and Instruction 19* (2009), 402–410.

[39] KALYUGA, S., AYRES, P., CHANDLER, P., AND SWELLER, J. The expertise reversal effect. *Educational Psychologist 38*, 1 (2003), 23–31.

[40] KOTLARSKY, J., VAN FENEMA, P. C., AND WILLCOCKS, L. P. Developing a knowledge-based perspective on coordination: the case of global software projects. *Information and Management 45* (2008), 96–108.

[41] KUHN, T. S. *The Structure of Scientific Revolutions*, 3rd ed. The University of Chicago Press, 1996.

[42] LATOZA, T. D., VENOLIA, G., AND DELINE, R. Maintaining mental models: A study of developer work habits. In *International Conference on Software Engineering* (May 2006), pp. 492–501.

[43] LYONS, E., AND COYLE, A., Eds. *Analysing Qualitative Data in Psychology*. Sage Publications, 2007.

[44] MAYER, R. E. *Multimedia Learning*, 2nd ed. Cambridge University Press, 2009.

[45] MOODY, D., AND VAN HILLEGERSBERG, J. Evaluating the visual syntax of uml: An analysis of the cognitive effectiveness of the uml family of diagrams. *Software Language Engineering* (2009), 16–34.

[46] MORRIS, M. G., SPEIER, C., AND HOFFER, J. A. An examination of procedural and object-oriented systems analysis methods: Does prior experience help or hinder performance? *Decision Sciences 30*, 1 (Winter 1999), 107–136.

[47] NUGROHO, A. *The Effects of UML Modeling on the Quality of Software*. PhD thesis, Leiden Institute of Advanced Computer Science, October 2010.

[48] NUSEIBEH, B., EASTERBROOK, S., AND RUSSO, A. Making inconsistency respectable in software development. *The Journal of Systems and Software 58* (2001), 171–180.

[49] OLSON, G. M., AND OLSON, J. S. Distance matters. *Human-Computer Interaction 15* (2000), 139–178.

[50] PAASIVAARA, M., AND LASSENIUS, C. Using iterative and incremental processes in global software development. In *The 3rd International Workshop on Global Software Engineering* (May 2004), pp. 42–47.

[51] PARNAS, D. L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM 15*, 12 (December 1972), 1053–1058.

[52] PERRY, D. E., STAUDENMAYER, N. A., AND VOTTA, L. G. People, organizations and process improvement. *IEEE 11*, 11 (July/August 1994), 36–45.

[53] PIDGEON, N. F., TURNER, B. A., AND BLOCKLEY, D. I. The use of grounded theory for conceptual analysis in knowledge elicitation. *International Journal of Man-Machine Studies 35* (1991), 151–173.

[54] POPPER, K. R. *The Logic of Scientific Discovery*. Rootledge Classics, 2002.

[55] PRIKLADNICKI, R., AUDY, J. L. N., AND EVARISTO, R. Global software development in practice. lessons learned. *Software Process: Improvement and Practice 8* (2003), 267–281.

[56] PRIKLADNICKI, R., AUDY, J. L. N., AND EVARISTO, R. An empirical study on global software development: Offshore insourcing of it projects. In *The 3rd International Workshop on Global Software Engineering* (May 2004), pp. 53–58.

[57] ROSSON, M. B., AND ALPERT, S. R. The cognitive consequences of object-oriented design. *Human-Computer Interaction 5* (1990), 345–379.

[58] SANDERS, K., BOUSTEDT, J., ECKERDAL, A., MCCARTNEY, R., MOSTRÖM, J. E., THOMAS, L., AND ZANDER, C. Student understanding of object-oriented programming as expressed in concept maps. In *SIGSE'08* (2008).

[59] SHRIVASTAVA, S. V., AND DATE, H. Distributed agile software development: a review. *Journal of Computer Science and Engineering 1*, 1 (May 2010), 10–17.

[60] TVERSKY, A., AND KAHNEMAN, D. Judgements under uncertainty: Heuristics and biases. *Science 185* (1974), 1124–1131.

About this research & me

My interest in the topic of Software Engineering founded a few years back during the homonymously named course. While I wanted to know more of this field, preferably through research, the idea for a thesis about the notational conventions of UML came to mind. Despite it being an interesting topic I abandoned it due to its large-scale (judging by the recently published paper of Moody and Hillegersberg, which is almost the research I had in mind [45]) and got side-tracked. At that time I followed several courses in cognitive psychology as well – and philosophy, but that's not the point here – that caught my attention. While my interest had always gone out for more than computer science alone, cognitive psychology proved to be the latent interest.

For my BSc thesis I wanted to examine the cognitive principles behind software architecture documentation (SAD), and I had the good fortune that a PhD student at LIACS was performing experiments that fitted these conditions. Subsequently I joined his experimentation and crystallised a research and analysis proposal meanwhile. It became the analysis of verbal protocols using Grounded Theory to gain insight into participant's minds – what are they doing to solve the proposed problem?

My knowledge of the mentioned methods of experiment and analysis was infinitesimal, so at least a little literature research into those fields was necessary to be able to apply them in a credible manner. Apart from this literature research, some motivation for this research – why would one want to know whether or not the current SAD fits human cognition? – appeared reasonable. My viewpoint here chosen was from the still upcoming practice of Global Software Development (GSD): due to impeded communication ways to reduce such difficulties should be found. I thought some liberation could be achieved through improvement of SAD and I still believe this to be true – now even with justified underpinnings.

Personally I gained much more than mere insight into the heads of our participants: performing literature research and executing an investigation and analysis was something I hadn't done before. Besides the obvious scientific value I want to emphasise this "non-functional" aspect: it was really worth doing it.

Appendix A1

Multimedia Learning

Overview

This appendix provides an overview of a current model of the human memory system and extends that model with the cognitive theory of multimedia learning by Mayer. Last, the implications for Software Architecture Documentation are discussed.

---

1. Introduction

"People learn better from words and pictures than from words alone". This statement is built on the hypothesis that learners are able to create a better internal representation using both media, a so-called *mental model*, which enables them to obtain a deeper understanding of the presented material. The most intriguing aspect of the qualitative rationale is that this understanding occurs when learners are able to build meaningful connections between verbal and pictorial representations.

The effects of multimedia learning are extensively researched by Mayer and colleagues and recently bundled in [15]. Mayer proposed the Cognitive Theory of Multimedia Learning to explain the effects of instructional design and hence multimedia learning. Section 2 will provide an outline of the cognitive memory systems, sections 3 and 4 will be based completely upon Mayers book and provide an introduction to the cognitive theory of multimedia learning and its implied guidelines for instructional design. Finally, in section 5 additional literature will be reviewed and implications for design documentations will be discussed.

2. Human understanding

At the basis of the cognitive theory of multimedia learning lies the way humans obtain, process, store and retrieve information. Extensive research in the area of cognitive psychology proposed several kinds of knowledge, several types of memory stores and theories about human attention and awareness, all in order to account for the complex human behaviour. This section will try to shed some light upon these intricacies of human learning, remembering and understanding.

2.1. Types of knowledge

Robillard, provides a clear overview of types of knowledge in his research into the role of knowledge in software development [20]. A common view nowadays subdivides knowledge into two main groups:

1. Procedural;
2. Declarative.

Procedural memory is implicit, and stores all information related to basic skills used to interact with our environment (e.g. walking, typing and cycling). Declarative memory on the other hand is stores explicit knowledge (e.g. facts and relations between objects). The second type is again dividable into separate groups: semantic and episodic knowledge. *Semantic knowledge* (sometimes also called encyclopaedic or topic knowledge) refers to definitions and meanings of words. The second type of declarative memory contains *episodic knowledge*, which consists of autobiographical events (like times, places, emotions and contextual knowledge).

2.2. Types of memory stores

2.2.1. Sensory stores

Perceptual information is briefly stored in modality-specific *sensory stores* and must be processed further to be able to reach conscious awareness. Sensory memory can be characterised by a large capacity and short duration of a few seconds. It includes the echoic and iconic memory types [12]:

1. *Echoic memory* is the auditory store of sensory memory. It can be divided into two stages: an ear-specific representation of the physical characteristics of the stimulus (referred to as an 'echo') and the combination of information obtained by both ears.
2. *Iconic memory* is the visual store of the sensory memory. It also has two stages: the retinal afterimage and a later stage in which information of both eyes is combined.

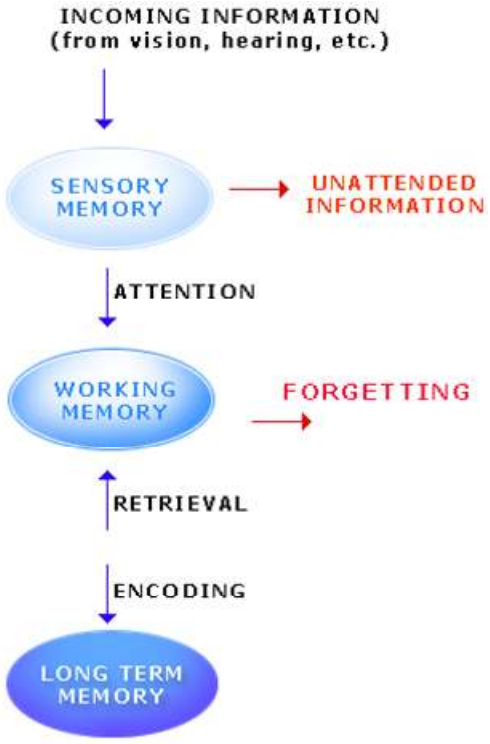


Figure 1: Human memory systems, adapted from [4: 2]

2.2.2. Working Memory

Information is held active in the *working memory* (WM), which is commonly supposed to be capable of processing both spatial and phonological information in respectively the visuo-spatial sketchpad and the phonological loop [12]. It generally refers to systems that are assumed to be necessary in order to keep things in mind while performing complex tasks such as reasoning, comprehension and learning [5]. The *visuo-spatial sketchpad* processes visual information whereas the *phonological loop* handles auditory and verbal information. The possession of a WM allows us to reflect and choose our actions, rather than instinctively react automatically to stimuli. Besides the phonological loop and the visuo-spatial

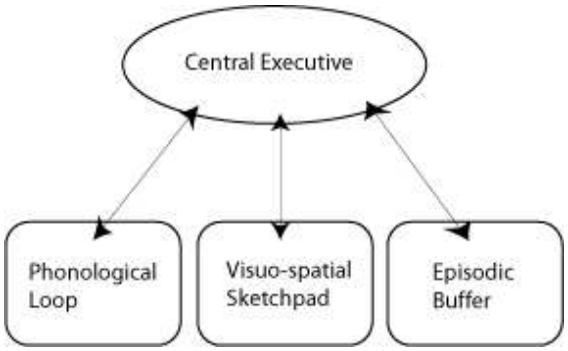


Figure 2: Model of the Working Memory, adapted from [5]

sketchpad, a controller called the central executive was introduced as well. The *central executive* is an all-purpose attentional controller and presumed to supervise and coordinate the work of its slave-systems [1]. In addition, relatively recently the existence of an *episodic buffer* was proposed. This buffer is assumed to facilitate integration of information in the phonological loop and the visuo-spatial sketchpad in a way that allows active maintenance and manipulation [2]. While in its initial form the buffer was assumed to play an active role in binding information, more recent research suggests that it serves as a rather passive store than an active processor [5 and references therein]. However, more research needs to be done on the workings of this component [3].

2.2.3. Long Term Memory

The *Long Term Memory* (LTM) consists of the memory types described in section 2.1: the procedural and declarative types. Converting information from WM to LTM is called *consolidation* [12]. This consolidation is supposed to take place in the form of *schema construction* or *knowledge construction* or simply *encoding* in **Error! Reference source not found.**. During this process, information is organised and structured into a schema (a knowledge construct) and linked to existing schemas residing in LTM.

2.2.4. Another model of memory

Of course the model of WM and LTM above is not the only one. For example, Cowan proposed a model in which WM constitutes an active part of the LTM [9]. The dissociation between Cowan's model, and Baddeley and Hitch's shed light on the problem from two different angles: Cowan derived his model from an initial focus on the psychological principle of attention, whereas Baddeley and Hitch were influenced by short-term verbal memory [5]. Nevertheless, Baddeley recognises: "[I]n practice, Cowan and I tend to agree with each other on most aspects (…) despite using very different theoretical metaphors" [5]. I will henceforth base the discussion of memory on the most used model of Baddeley and Hitch.

2.3. Attention and awareness

Experts are better at what they do because they have ways to diminish the cognitive load needed in order to execute a task or engage in problem solving. Previous experience delivered them schemas that enable them to depend more on LTM, thus freeing up WM resources for other tasks [12: 282]. Increasing one's level of expertise in a domain is a major means of reducing WM load [14]. However, at the downside, one has a decreased situation awareness. *Situation awareness* can be described as "being aware of and understanding both the current situation and the way in which it is evolving, such that appropriate decisions can be made actions taken" [12: 280]. Loss of situation awareness due to over reliance on poor, incorrect or inappropriate schemas, generally increases the amount of errors made [12: 282].

2.4. Conclusion: how do humans understand?

A conclusion cannot be put as indisputable as might be expected from the evidence above: the social sciences have not come up with a conclusive account yet. Nevertheless, we do know that experience plays a major role in any knowledge-related activity. At the basis of experience lies the attainment of information and the appropriate organisation for simple retrieval. Believed is that the organisation and integration phases (described below) account for human understanding of problems and enables them to retrieve suitable schemas to apply when encountering new situations [15]. Understanding occurs as a result of selecting the right information and combining it with apt existing knowledge [13].

3. Cognitive theory of multimedia learning

At the basis of the cognitive theory of multimedia learning stands the idea that the design of multimedia messages should be consistent with the way people process information. This 'way' is based upon three principles of the cognitive science:

1. Dual-Channel assumption;
2. Limited Capacity assumption;
3. Active Learning assumption.

In short, the *Dual-Channel* assumption is the separation of channels for processing visual and auditory information; the *Limited Capacity* assumption assesses the limitation on the amount of information humans can process in a channel at a time; and the *Active Learning* assumption refers to the idea that humans need to construct knowledge from presented material rather than just to absorb it.

The integration of verbal and pictorial models is crucial in successful multimedia learning. This can be equated with the Active Learning-assumption and could be subdivided into 3 moments:

1. Selecting;
2. Organising;
3. Integrating.

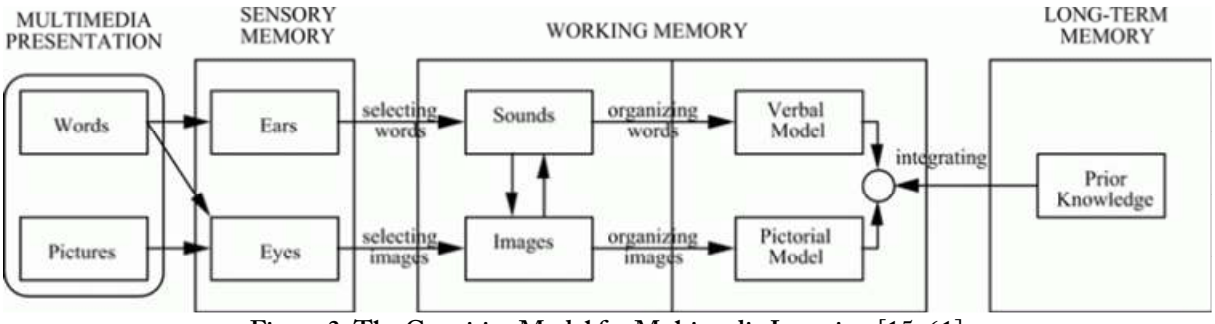


Figure 3: The Cognitive Model for Multimedia Learning [15: 61]

Each of the annotated arrows in figure 3 is linked to the mentioned processing stages. Information enters the pathway through the eyes or ears in the sensory memory, and subsequently the first stage of learning/understanding commences: first, one selects interesting information out of all presented stimuli. Second, the learner constructs separated verbal- and

pictorial models by creating connections between the obtained verbal elements and between the obtained pictorial elements. Last, the models built are integrated into one mental model.

As described above, learning requires cognitive processing and hence induces cognitive load. Three kinds of cognitive load are to be discerned:

1. Extraneous Processing;
2. Essential Processing;
3. Generative Processing.

The first type encloses cognitive processing that does not serve the instructional goal. *Essential Processing* is required to represent the essential material in working memory and *Generative Processing* is required to create a deeper understanding of the presented materials. In order to ameliorate the process of selecting, organising and integrating, instructional designers need to reduce extraneous processing, manage essential processing and foster generative processing. All principles described in section 4 will serve one or more of these goals.

4. Principles

1. Coherence

People learn better when extraneous information is excluded rather than included. This works because learners are actively trying to make sense of the presented material and adding extraneous information gets in the way of this structure building process. The implications of this principle for multimedia design are clear: do not add extraneous words or pictures in a multimedia presentation. Needed elaboration should only be presented *after* the learner has constructed a coherent mental representation of the concise cause-and-effect system.

2. Signaling

People learn better when the essential material is highlighted. The signals are intended to guide the learner towards essential material and to help the learner organise this information into a coherent structure. Evidence for the signaling principle is promising but still preliminary only shows a medium effect. Signals should be used sparingly, with a maximum of 1 or 2 per paragraph. Nevertheless, current research only provides evidence for verbal signaling; not for visual signaling.

3. Redundancy

People learn better from graphics and narration than from graphics, narration *and* written text. This principle builds on the idea that concurrently using these media induces cognitive overload: people try to compare all incoming information streams and hence fail to retain adequate cognitive resources to integrate all information.

Implications are clear: do not add concurrent written text to narration and graphics. However, when the redundant text consists of only a few words, the effect of this principle vanishes. Moreover, learning is improved when the narration is presented first and written and pictorial materials afterwards.

4. Spatial Contiguity

People learn better when corresponding information is presented near rather than separated far from each other. This principle builds on the idea that presenting corresponding information close to each other reduces the need for visual search. The learner is then better able to hold the information in his WM, hence retaining cognitive capacity that can be used to integrate information. The spatial contiguity principle works best when …

1. … the learner is not familiar with presented information.
2. … the diagrams need words to be understood.
3. … the material is complex.

5. Temporal Contiguity

People learn better when corresponding words and pictures are presented simultaneously rather than successively. When both media are presented concurrently, the learner is better able to hold presentations of both media in his WM, hence improving their ability to build mental connections between the verbal- and pictorial representations.

6. Segmenting

People learn better when the information is presented in user-paced segments than in a continuous stream. This will allow the user to complete each step in the process of understanding, from selection to integration at their own pace. Segments should be meaningful and have to serve a clear sub goal.

7. Pre-training

People learn more deeply from a multimedia message when they know the names and characteristics of the main concepts. People then have the opportunity to build the component model first and subsequently a causal model. A component model consists of the name and behaviour of each component; the causal model consists of causal chains between those components. This principle works because these two essential processes in learning are divided and presented successively. Ensuring that learners possess appropriate knowledge beforehand helps to prevent cognitive overload in essential processing.

8. Modality

People learn more deeply from spoken text and pictures than from printed text and pictures. This principle positively affects learning by off-loading the visual channel when text is communicated via the verbal channel. The modality effect applies the strongest when understanding is required rather than mere retention.

On the downside, this principle may not apply when people experience difficulty in finding corresponding diagram-parts to the narrated text, but signaling might be a solution to this. Also, printed text may be favoured over narration when using many jargon terms, when the learners are non-native speakers, hearing-impaired or when the lesson contains hard to pronounce words and symbols.

9. Multimedia

People learn better from pictures and words than from words alone. Words and pictures are 2 qualitatively different systems for the representation of knowledge. The act of building connections between those mental models is an important step in conceptual understanding, thus fostering generative processing.

Complementary research on the multimedia principle is research on *graphic advance organisers*. They usually consist of graphics and text and foster understanding, especially by priming prior knowledge that can be integrated into the incoming message.

10. Personalisation

People learn better when words are presented in a conversational manner rather than in a formal style. People try harder to understand what the author is saying when they perceive as if the author is talking with them. Also, people tend to understand information better when the narrating voice is human rather than mechanical, especially when the narrator is similar to the learner self in terms of gender, race, ethnicity and emotional state. However, more research is needed to establish guidelines on how to add personalisation without overdoing it.

4.1. General boundary conditions

All of these principles require further research to find out more about their applicability. In general, the principles described above have the following boundaries:

1. They apply best to low-skilled and low-experienced learners. Sometimes they even invoke what is called the Expertise Reversal Effect;
2. The presentation of information is fast-paced rather than low-paced or user-paced.

High-knowledge learners may be able to create pictorial representations by themselves, therefore being hindered by integration of pictures and text: high-knowledge learners learn best using pictorial messages only. More information on this Expertise Reversal Effect can be found in section 5.2.2.

The second boundary condition can be explained by the effects of cognitive overload. When the presentation of information is rather fast-paced, the learner may not be able to process all necessary material and integrate it into one mental model. User-paced presentations enable the learner to finish a complete sequence of selecting, organising and integrating before moving on to the next segment.

5. Implications for SADs

5.1. A change of paradigm

The implications of these principles for Software Architecture Documentation are manifold. First, it is clear that a paradigm shift from the information-delivery view[1] to the knowledge-construction view is required in order to see the problems and their solutions: it is not only the completeness, style and rigor of design documents that results in improved communication, but clearly the way it is tried to convey as well. The designers have to think about how developers extract meaning from their documents (psychophysics), what people understand from it (cognition), and how it is imbued with meaning (semiotics) [10]. In order for humans to understand and use information to solve problems and make good programming decisions, they need at least a high-quality mental model which can only be built when the presentation of information allows for the construction of such a rich model.

The creation of a rich mental model takes effort and produces understanding of the system at hand: hence, it would be practical to find a way of storing that information into generally accessible media. Almost automatically, this leads us back to SADs: while understanding – schema construction and subsequent schemas – is subjective[2], one could still try to fit documentation to an average schema. Furthermore, if documentation facilitates easy storage – by better tooling – and simple retrieval – by applying the principles of Multimedia instruction – developers might be more inclined to use it as a vehicle of useful and up-to-date information about the system.

5.2. Related research

5.2.1. Diagrams

Related research on comprehension of UML class diagrams has been performed with the use of an eye-tracker [22]. Yusuf *et al.* found that, compared to novices, experts tend to make more use of stereotype information as colouring and textual annotations, and use the layout to

[1] Within the information-delivery view, humans are seen as single-channel, unlimited capacity, passive-processing systems [15: 61].

[2] Note the difference between information and knowledge: even a record of a mental model does not consist of knowledge, as that is only something that can reside in someone's head. A written down mental model merely consists of the components and behaviour, and the connections in between them.

explore and navigate through the class diagrams. Specifically, they state that experts have a tendency to navigate from the centre to the edges. Novices on the other hand, read the diagram top-down and left-to-right, without taking into account any of those signals. Other research has shown that advanced learners may be able to immediately see the higher-level structures in material while novices could only see random low-level components [13]. This indicates that experienced individuals tend to use a top-down approach, grouping problems based on their underlying structure, while novices use bottom-up approaches and hence group sets of problems based on surface features [16] and references therein].

Other research on model comprehension came up with the conclusion that less intuitive notations lead to improved error detection [19]. Purchase *et al.* ascribe this finding to the idea that their participants might be less at ease with an unintuitive notation, hence increasing diligence in seeking for mistakes. Hadar and Leron draw a matching conclusion from their research: "Intuition is a powerful tool, which helps us navigate successfully through most everyday tasks. However, it may at times get in the way of more formal processes." [11].

Cherubini *et al.* did some research on the extend to which drawings are used [8]. They found that most drawings were informal and transient (i.e. they were lost after a meeting finished, so they had no later value) and that they were mostly used to increase understanding, to design and to communicate with others. In their results, the size of a box sometimes intuitively encoded the importance of the entity it represented. Also, relationships were indicated with arrows (instead of arrowless lines in UML), and related components were drawn together as a group (an example of the *Spatial Contiguity* principle).

5.2.2. Cognitive psychology

Research closely related to the cognitive theory of multimedia learning, was carried out by Butcher [6]. Her investigation upon the potential effects of different diagram representations resulted in a differentiation between levels of detail: although more abstract diagrams are usually more difficult to understand, they supported learners to generate more integration inferences. Moreover, those inferences are more correct as well. Participants using diagrams engage in more useful comprehension processes more frequently. Furthermore, she states that a reduction of a diagram's representational complexity (i.e. application of the *Coherence* principle) in the service of highlighting structural relationships support understanding. In another recent study by Butcher, she investigated upon the effects of interactive diagrams in learning [7]. The outcomes speak volumes: diagram interaction resulted in even better transfer

performance. Additionally, an increased amount of deep cognitive processes like self-explanations was invoked.

Paas *et al.* have shed some light upon the cognitive load aspect [18]. In their search for literature about cognitive load, they found astounding amounts of literature that supports the main hypothesis behind the cognitive theory of Multimedia Learning [15]: "Differences in effectiveness between instructional formats are largely based upon differences in memory load" [18] and references therein]. Furthermore: "[P]erformance has been shown to degrade by either underload or overload".

Corresponding research by Kalyuga *et al.* has been done upon the Expertise Reversal Effect and knowledge elaboration [13, 14]. They identified the negative consequences of instructional guidance – like the principles described in section 4 – and identified a cause. While instructional guidance for novices acts as a substitute for missing schemas from LTM and as a means of constructing new schemas, for experts it only interferes with their existing schemas. Experts have a rich collection of different schemas that could readily be applied to a situation. Overlap between the pre-existent schemas and provided instructional guidance will induce cross-referencing between the two. Like the effects of improper application of the *Redundancy* principle, this extra process might cause a cognitive overload and hence thwart learning [14].

5.2.3. Object-Oriented Development

Research about the cognitive consequences of Object-Oriented Development (OOD) on problem understanding has been carried out by Rosson and Alpert [21]. They suggest that "[t]he problem understanding achieved must be one whose structure suggests a solution plan" and indicate that a design solution that "can be built on top of existing structures in memory (…) will be easier to maintain". This is assessed by the great benefit OOD brought: "Perhaps the greatest strength of an object-oriented approach to development is that it offers a mechanism that captures a model of the real world." [21] and references therein]. More information about OOD can be found in Appendix A2.

5.3. Practical implications

The combination of empirical research in software engineering and a theoretical framework – which is based on empirical work as well, for that matter – of cognitive psychology constitutes a strong fundament upon which to build. Several guidelines can be constructed:

- Research on diagrams showed that emphasising crucial elements might help developers in assessing their importance. This could be achieved using the *Signaling* principle, by increasing the size relatively to importance or by using colours to assign importance [8], and the *Coherence* principle to strip the diagrams from extraneous materials [6].
- A central placement of important elements seems to be recommended.
- The *Spatial contiguity* principle should be used to group similar or highly coupled elements [15].
- The *Segmenting* or *Pre-training* principles should be applied in order to separate the essential processes of understanding into multiple sessions at user pace: first obtain the names and behaviours of components separately and subsequently their relations [15]. This is also partly supported by Butcher, who recognises that visual summaries – diagrams – may prompt learners to integrate information and provide cues for recall [6].
- Following the boundary condition of the *Redundancy* principle and the idea behind the *Pre-training* principle, the offering of a spoken overview of the system first and subsequently the written documentation and text increases understanding. This could be pushed into practice by design exploration meetings with designer and developers together, applying a literal version of the *Personalisation* principle.
- Appropriately measuring levels of knowledge per developer in real time is important to effectively balance the executive external guidance [13]. Another, perhaps easier realisation might be a tool with an adaptive Level of Detail (LoD). Supporting different LoDs enables a developer to adjust the documentation to his needs. For novices, then even a record of common ground[3] could be implemented at the lowest LoD.

[3] *Common ground* refers to the knowledge that different people have in common and that they are aware of the fact they do. Common ground is easily established when collocated, because developers then not only share cultural and linguistic backgrounds, but then there also exists better awareness about the microcontext – what one's doing, what remains to be done et cetera [17]. Common ground is also known as *transactive memory*.

- Consistency in SADs might be appointed using dynamic signaling: when a developer selects a component in the diagram, accompanying text should be selected as well. This eases the integration-stage and as a result facilitates consistency checks by the developer. In addition, while the designer should introduce the relations between documents explicitly, he is forced to check consistency as well.
- Apart from highlighting relations across documents, relations and dependencies within one document should be dynamically signalled as well: with the selection of one component, all related components should become highlighted to increase visibility of dependencies.

Besides the above mentioned guidelines, it are not only the cognitive advantages that can be exploited to a further extend. For example, increased formalisation of a graphical language paves the way for more extensive type checks [10]. More information about this can be found in Appendix A2.

Of course the list of ideas expounded above is by no means exhaustive. Following research on cognition and combining it with the domain of software development, much more is to be gained by the improvement of software architecture documentation.

6. References

[1] BADDELEY, A. Working memory. *Life Sciences 321* (1998), 167–173.

[2] BADDELEY, A. The episodic buffer: a new component of working memory? *Trends in Cognitive Science 4*, 11 (November 2000), 417–423.

[3] BADDELEY, A. D. Is working memory still working? *European Psychology 7*, 2nd (June 2002), 85–97.

[4] BADDELEY, A. D. *The Essential Handbook of Memory Disorders for Clinicians*. John Wiley and Sons, 2004.

[5] BADDELEY, A. D. Working memory. *Current Biology 20*, 4 (February 2010), 136–140.

[6] BUTCHER, K. R. Learning from text with diagrams: Promoting mental model development and inference generation. *Journal of Educational Psychology 98*, 1 (2006), 182–197.

[7] BUTCHER, K. R. How diagram interaction supports learning: Evidence from think alouds during intelligent tutoring. *Diagrammatic Representation and Inference* (2010), 295–297.

[8] CHERUBINI, M., VENOLIA, G., DELINE, R., AND KO, A. J. Let's go to the whiteboard: how and why software developers use drawings. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (April-May 2007), pp. 557–566.

[9] COWAN, N. *Working Memory Capacity*. New York: Psychology Press, 2005.

[10] DUKE, D. J., DUCE, D. A., AND HERMAN, I. Do you see what i mean? *IEEE Computer Graphics and Applications 25*, 3 (May/June 2005), 6–9.

[11] HADAR, I., AND LERON, U. How intuitive is object-oriented design? *Communications of the ACM 51*, 5 (May 2008), 41–46.

[12] JOHNSON, A., AND PROCTOR, R. W. *Attention: Theory and Practice*. SAGE publications, 2004.

[13] KALYUGA, S. Knowledge elaboration: A cognitive load perspective. *Learning and Instruction 19* (2009), 402–410.

[14] KALYUGA, S., AYRES, P., CHANDLER, P., AND SWELLER, J. The expertise reversal effect. *Educational Psychologist 38*, 1 (2003), 23–31.

[15] MAYER, R. E. *Multimedia Learning*, 2nd ed. Cambridge University Press, 2009.

[16] MORRIS, M. G., SPEIER, C., AND HOFFER, J. A. An examination of procedural and object-oriented systems analysis methods: Does prior experience help or hinder performance? *Decision Sciences 30*, 1 (Winter 1999), 107–136.

[17] OLSON, G. M., AND OLSON, J. S. Distance matters. *Human-Computer Interaction 15* (2000), 139–178.

[18] PAAS, F., TUOVINEN, J. E., TABBERS, H., AND VAN GERVEN, P. W. M. Cognitive load measurement as a means to advance cognitive load theory. *Educational Psychologist 38*, 1 (2003), 63–71.

[19] PURCHASE, H. C., COLPOYS, L., MCGILL, M., CARRINGTON, D., AND BRITTON, C. Uml class diagram syntax: an empirical study of comprehension. In *APVis '01: Proceedings of the 2001 Asia-Pacific symposium on Information visualisation* (Darlinghurst, Australia, Australia, 2001), vol. 9, Australian Computer Society, Inc., pp. 113–120.

[20] ROBILLARD, P. N. The role of knowledge in software development. *Communications of the ACM 42*, 1 (January 1999), 87–92.

[21] ROSSON, M. B., AND ALPERT, S. R. The cognitive consequences of object-oriented design. *Human-Computer Interaction 5* (1990), 345–379.

[22] YUSUF, S., KAGDI, H., AND MALETIC, J. I. Assessing the comprehension of uml class diagrams via eye tracking. *International Conference on Program Comprehension 15* (June 2007), 113–122.

Appendix A2

Object-Oriented Development

Overview

This appendix constitutes an introduction to Object-Oriented Development (OOD). As it is the method used in our experiments, some basic knowledge about its history, features, alleged merits and complexities helps getting the bigger picture. Also, solutions to mitigate some of the inherent complexities will be proposed.

---

1. Introduction

1.1. What is Object-Oriented Development? Its characteristics.

There exist five types of programming styles that are fundamentally different from each other [2: 37]:

- Procedure-oriented (which is structured around algorithms)
- Object-oriented (uses classes and objects)
- Logic-oriented (often expressed using predicate calculus)
- Rule-oriented (with the use of mere If-then clauses)
- Constraint-oriented (based upon invariant relationships)

The orientation under investigation here, Object-Oriented Development (OOD), has several distinguishing properties and concepts that make it a unique approach software development [1]:

- *Inheritance*: A mechanism that allows the data and behaviour of one class to be included in or used as the basis for another class.
- *Object*: An individual, identifiable item, either real or abstract, which contains data about itself and descriptions of its manipulations of the data. It has a state, behavior, and identity [2: 81].
- *Class*: A description of the organisation and actions shared by one or more similar objects.

- *Encapsulation*: A technique for designing classes and objects that restricts access to the data and behaviour by defining a limited set of messages that an object of that class can receive.
- *Method*: A way to access, set or manipulate an object's information.
- *Message Passing*: The process by which an object sends data to another object or asks the other object to invoke a method.
- *Polymorphism*: The ability of different classes to respond to the same message and each implement the method.
- *Abstraction*: The act of creating classes to simplify aspects of reality using distinctions inherent to the problem.

Although the list of concepts above is by no means the only one constructed, Armstrong *et al.* claim they did capture all the *important* facets of OOD. Despite the wide usage of OO-approaches, there still is both a lack of consensus on its fundamental concepts and the understanding how they can be classified to characterise the OO approach. This is found very confusing, especially for the developer trying to master the OO approach [1].

Because the decomposition of the problem space is based upon objects and not algorithms, this is called an *object-oriented* approach or decomposition [2: 16]. Hence, central to OOD is the metaphor of communicating objects [22].

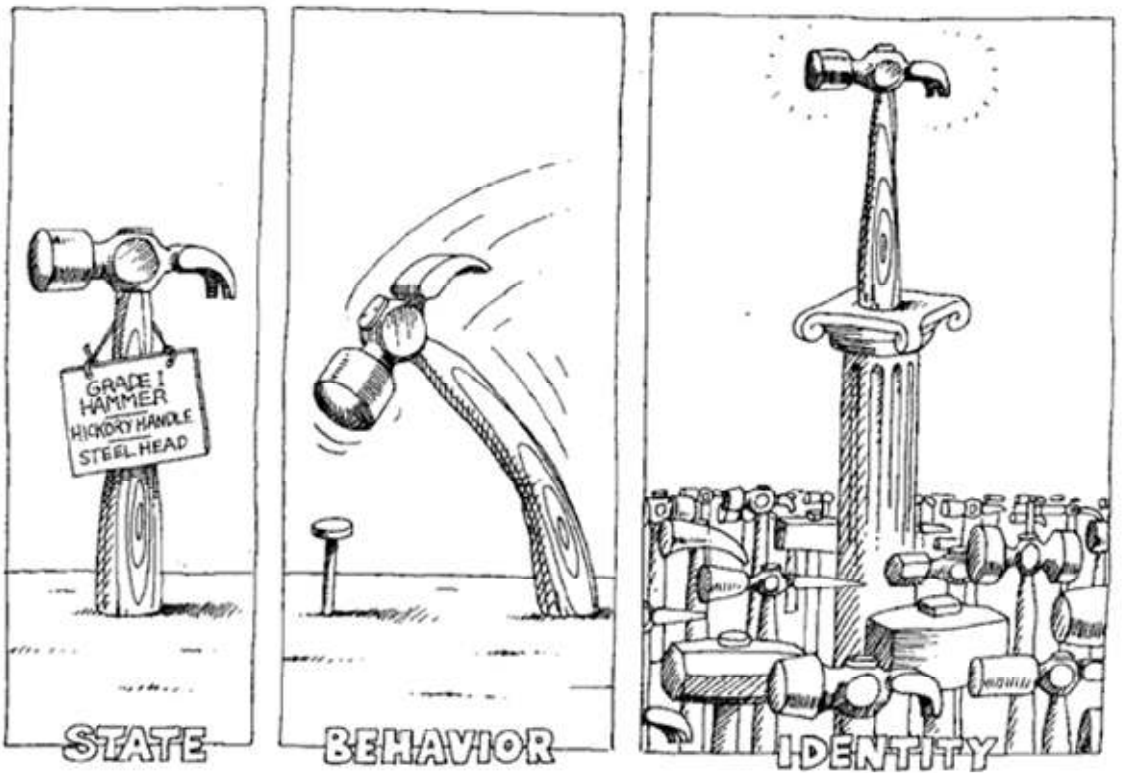


Figure 1: Properties of an object: State, Behaviour and Identity [2]

A state of an object encompasses the static properties and dynamic values of it. The behaviour is how an object acts and reacts on other objects through *Message Passing*. It is a way of manipulating the structure of itself or other objects: a state changes occur as a result of behaviour [1]. The identity is the property that distinguishes an object from all other objects [2: 82-89].

Some have called the OO-approaches a new paradigm for software development [19 and references therein]. It is different from other types of programming languages: the focus is on *what* rather than on *how*. Especially compared to Procedure-oriented development, OOD starts off more high level with structure, filling in the technical details at later stages [22]. This is fully compliant with the recognition of Brooks that software entities should be grown instead of built. According to him, systems should first be made to run and subsequently be fleshed out [3].

2. The evolution of Object-Oriented Development

2.1. Its introduction

OO was introduced in the late 60s with Simula 67 [1, 2: 34]. The propagation of this new approach took form due to several events back then [2: 33 and references therein]:

1. Advances in computer architecture starting over 35 years ago with the development of hardware support for operating system concepts;
2. Advances in programming languages, as demonstrated with Simula, Smalltalk, CLU, and Ada;
3. Advances in programming methodology, including modularization and Information Hiding [20];
4. Advances in database models (through the Entity-Relationship approach);
5. Research in Artificial Intelligence.

2.2. Alleged merits

The intention with the creation of the Object-Oriented paradigm was to deal with the ever-increasing complexity of software systems. The idea was to exploit the human mind's natural capabilities for thinking about the world in terms of objects and actions [11], and indeed, research has shown that novices tend to prefer OO-techniques over Procedural techniques [19]. Although all subjects in this study by Morris *et al.* shown a higher cognitive load when using OO-techniques, it seems that OO-techniques are simpler because they fit the cognitive functions of humans to a greater extend [2: 74-76]:

- Because a design schema represents a relatively stable structure comparable to a cognitive schema in LTM[1], it serves as an organising function, supporting storage and retrieval of intermediate solutions. If a design solution can be built on top of such existing structures, it will be easier to maintain as it is further defined and tested. Also, it will be more robust when a design becomes more complex [22 and references therein].

[1] A *cognitive schema* or knowledge construct consists of organised and structured information in LTM. In addition, it is linked to other existing schemas residing in LTM, forming some sort of semantic network.

- OO-designing allows a designer to generate an initial design model in the context of the problem itself, rather than requiring a mapping onto computer science constructs. It supports a better integration of problem and solution: designers can simply devote more attention to problem understanding rather than technical details [22]. For example, objects often correspond to "real world" items from the problem domain instead of prescribed constructs [19].
- Objects can represent separable chunks which simplifies the mental representation [22, 20]. It helps to subdivide a big system into smaller parts which are simple enough to be understood [7].

Besides these cognitive advantages, reusability of design increases development pace, quality and flexibility and hence decreases cost [2: 74-76, 22, 20].

2.3. Complexities

2.3.1. … of software development in general

Software entities are recognised to be far more complex for their size than perhaps any other human construct: no two parts are alike. Furthermore, scaling up a software entity necessarily results in an increase of the number of distinct elements. Greater project sizes come with enlarged difficulty of communication which is a part of the complexity of software development as well [14, 3]. Much complexity comes from conformation to other interfaces.
Nevertheless, the biggest hindrance is the absence good metaphors for software due to its non-spatial nature: the structures of software remain inherently unvisualisable, hence impeding the process of design in one mind and communication of it among minds. Unfortunately, these complexities are regarded as essential rather than accidental: the essence of the software *are* its complexities (such as communication and integration between different components) [3].

2.3.2. … of Object-Oriented Development

Developers with previous experience in Procedural development encounter impediments of OOD: a designer's past experience unavoidably influences the ease of generating object-based problem representations [22]. A stumbling block to obtaining the benefits of OO is learning the approach, regardless any previous experience [19]. In research by Morris *et al.* both novice and subjects experienced with Procedural development methods demonstrate higher Subjective Mental Workload (SMW) when using OO [19]. Nevertheless, novices prefer OO-techniques

over Procedural techniques and find these techniques easier as compared to users experienced with Procedural methods [19].

The construction of a design solution based on structures in LTM eases its maintenance and improves robustness. Besides, our cognitive system certainly makes extensive use of objects and categories in real life [11]. Nevertheless, individuals may construct invalid analogies, using their prior experience inappropriately [19], or may not even possess the relevant design schemas [22 and references therein]. Moreover, while objects, classes and inheritance certainly have an intuitive flavor, their formal version in OOD differs in important ways from their intuitive origins [11].

2.3.3. …of communicating a design

In addition to the complexities of the design and development itself, there is the difficulty of communicating a design with all stakeholders. Such communication is of critical importance as it has considerable influence on the success of a development project [16, 12, 13, 17]: it provides the only possibility that the separate task groups will be able to consolidate their efforts into a unified system [7].

Designs are subdue to interpretation: both the designer and the programmer need to understand the notational convention and use it accordingly in order to transfer knowledge effectively. Drawings might be the standard way of communication in other engineering areas, in computer science they are not: code is king [18]. This results in a tendency to adopt informal, *ad hoc* notations [5]. However, for a project to be successful, common representational conventions and terminology need to be established to facilitate effective communication [16, 8].

Both formal and informal interpersonal communication is valuable in software development [17]. However, in the light of GSD, there is a lack of informal and unplanned "coffee machine talk" [21], which increases the burden on the formal methods of communication like (design-) documentation. Because the nature of software requirements and design is to be unstable – all successful software gets changed [3] – documents tend to get out-of-date easily [16, 4, 5, 15, 14, 8].

3. Applying Object-Oriented Development

Object-Oriented Development has several benefits over the traditional Procedural Development. However, software development remains to be a creative task in which the relative ease of solving a problem will depend on how successful the solver has been in representing critical features of the task environment in his problem space [19 and references therein, 3]. As Object-Oriented programming can only remove accidental complexities, it will not be the "silver bullet" to the ever increasing complexity in the software venture [3]. Evidence in the previous section suggests that the OOD paradigm could be advantageous but should not merely be applied. There exist some marginalia.

3.1. Notation

First, although the idea to use objects and actions in a development paradigm seems fruitful, the slight modifications of the basic elements of the paradigm (like objects, classes and inheritance) as compared to the intuitive application appears detrimental. Some gain could be obtained using minor adjustments in notation – like UML.
In addition, at least a modest form of formality is advisable here: because software architecture documents (SAD) are used in collaborative activities (i.e. software development), a shared graphic vocabulary and hence shared meaning and usage of its components is required to come to the best results. Furthermore, formalisation eases automated checks like type-checking [10].

3.2. Style

Second, apart from notational modifications, modeling style could be adjusted as well. As mentioned before, designs are subdue to interpretation: differences in intention of the designer and the interpretation of the programmer result in a gap between model and code which probably leads to problems at integration and may ultimately result in a failed software project. In order to decrease the risk of misunderstandings and ambiguities in communicating a design, principles coined by research in cognitive psychology could be applied. More information about these principles and its implications for software development can be found in Appendix A1.

3.3. Ease of documenting

Third, tooling should be employed in order to elicit documentation of both *explicit* and *implicit* design decisions. "Lots of design information is kept in people's heads", thus never written down nor shared amongst people [18]. For example, designers mostly use free-form notations with *ad hoc* semantics to capture ideas while brainstorming, while a formal modeling language like UML was only used at a later stage. Tools supporting software design therefore should support different levels of formality, which is currently not the case [5, 9]. In addition, as software development is rarely a fully predictable undertaking, it should accomodate easy changes and means to make those changes noticed by the developers involved as well.

3.4. Using design documentation

Fourth, the cost of *using* design documents should be reduced: an increased likelihood that the right document can be found, better capture of informal, transient designs, and a decrease in necessary effort to keep them up-to-date should be somehow implemented in new tools [18]. If documents are always up-to-date and easy to find, they could become a way of communication [18, 6].

4. Conclusion

Intuition is a powerful tool, which helps us navigate successfully through most everyday tasks, but may get in the way of more formal processes like the development of software [11]. OOD made a step into a direction where the power of our built in system for using objects and actions is exploited. However, the solutions raised in the sections above are just appealing to the surface-features of the apparent difficulties. More research is needed on why it does not always alleviate the problems encountered, and what could be done to improve and ease the processes intrinsic to software development. Where Brooks thought that "[t]he most important single effort we can amount to is to develop ways to grow great designers", we think that there is much to be gained from improving the development methods and means itself as well [3].

5. References

[1] ARMSTRONG, D. J. The quarks of object-oriented development. *Communications of the ACM 49*, 2 (February 2006), 123–128.

[2] BOOCH, G. *Object Oriented Analysis and Design. With applications*, 2nd ed. Addison Wesley Longman, 1994.

[3] BROOKS, F. P. No silver bullet: Essence and accidents of software engineering. *IEEE Computer 20*, 4 (1987), 10–19.

[4] CATALDO, M., BASS, M., HERBSLEB, J. D., AND BAS, L. On coordination mechanisms in global software development. In *International Conference on Global Software Engineering* (2007), pp. 71–80.

[5] CHERUBINI, M., VENOLIA, G., DELINE, R., AND KO, A. J. Let's go to the whiteboard: how and why software developers use drawings. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (April-May 2007), pp. 557–566.

[6] CHISAN, J., AND DAMIAN, D. E. Towards a model of awareness support of software development. In *The 3rd International Workshop on Global Software Engineering* (May 2004), pp. 28–33.

[7] CONWAY, M. E. How do committees invent? *Datamation 14*, 1 (April 1968), 28–31.

[8] CURTIS, B., KRASNER, H., AND ISCOE, N. A field study of the software design process for large systems. *Communications of the ACM 31*, 11 (November 1988), 1268–1287.

[9] DEKEL, U. Supporting distributed software design meetings: what can we learn from co-located meetings? In *Proceedings of the 2005 workshop on Human and social factors of software engineering* (2005), pp. 1–17.

[10] DUKE, D. J., DUCE, D. A., AND HERMAN, I. Do you see what i mean? *IEEE Computer Graphics and Applications 25*, 3 (May/June 2005), 6–9.

[11] HADAR, I., AND LERON, U. How intuitive is object-oriented design? *Communications of the ACM 51*, 5 (May 2008), 41–46.

[12] HERBSLEB, J. D. Global software engineering: The future of socio-technical coordination. In *FOSE'07* (2007).

[13] HERBSLEB, J. D., AND GRINTER, R. E. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st International Conference on Software Engineering* (1999), pp. 85–95.

[14] HERBSLEB, J. D., AND MOCKUS, A. An empirical study of speed and communication in globally distributed software development. *IEEE transactions on Software Engineering 29*, 6 (June 2003), 481–494.

[15] HERBSLEB, J. D., PAULISH, D. J., AND BASS, M. Global software development at siemens: experience from nine projects. In *Proceedings of the 27th international conference on Software engineering* (May 2005), pp. 524–533.

[16] KOTLARSKY, J., VAN FENEMA, P. C., AND WILLCOCKS, L. P. Developing a knowledge-based perspective on coordination: the case of global software projects. *Information and Management 45* (2008), 96–108.

[17] KRAUT, R. E., AND STREETER, L. A. Coordination in software development. *Communications of the ACM 38*, 3 (March 1995), 69–81.

[18] LATOZA, T. D., VENOLIA, G., AND DELINE, R. Maintaining mental models: A study of developer work habits. In *International Conference on Software Engineering* (May 2006), pp. 492–501.

[19] MORRIS, M. G., SPEIER, C., AND HOFFER, J. A. An examination of procedural and object-oriented systems analysis methods: Does prior experience help or hinder performance? *Decision Sciences 30*, 1 (Winter 1999), 107–136.

[20] PARNAS, D. L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM 15*, 12 (December 1972), 1053–1058.

[21] PERRY, D. E., STAUDENMAYER, N. A., AND VOTTA, L. G. People, organizations and process improvement. *IEEE 11*, 11 (July/August 1994), 36–45.

[22] ROSSON, M. B., AND ALPERT, S. R. The cognitive consequences of object-oriented design. *Human-Computer Interaction 5* (1990), 345–379.

Appendix A3

Global Software Development

Overview

This appendix is an introduction in what Global Software Development (GSD) is, why we use it and what the problems are with geographically separated development locations. Next, we will sketch a future of GSD with some best practices and possible technologies that might become available. Finally, we come up with the implications that GSD has for Software Architecture Documents.

---

1. Introduction

The process of globalisation in parts of the economy other than software development began back in the early 19th century. It is therefore a very modern phenomenon with many unknown aspects [33]. The globalisation of software development started off around the 60s and 70s [29], but began its real sprint only about ten years ago [14]. Unfortunately, most of the incentives for both outsourcing[1] and offshoring[2] are still not well founded and the understanding of the effects it produces resides therefore still in its infancy: practice appears to be ahead of research as yet [30, 1]. Both successes and failures are experienced with GSD, but not very much is known about the reasons for this contrast [38, 32]. Despite this, outsourcing was incorporated in many organisational strategic plans, in both large firms with mature IT departments [29] and small to medium sized companies [5]. As a result, software development is becoming a multi-site and multicultural undertaking, impacting

[1] *Outsoucing* is the transferral of certain parts of software development to third-party service providers. I will use the term outsourcing instead of offshored outsourcing.

[2] A company can *offshore* when it has multiple departments in different countries. Contrary to outsourcing, this kind of displaced development is still in-house. I will use the term offshoring instead of offshored insourcing. Both offshored outsourcing and offshored insourcing are handled the same in this appendix, since the two are equivalent for most pragmatic purposes [2].

not only marketing and distribution, but also the way products are conceived, designed, constructed and tested [38] and references therein].

2. Current state of practice and research

The search for competitive advantages often forces companies to look abroad for cost reduction, quality enhancement, flexibility increase, risk dilution, and productivity improvement [39]. Globally distributed software development has many potential benefits but brings lots of drawbacks as well [4, 11, 38, 3]. Moreover, the assumed benefits that drive companies to go global turn out to be not that overwhelming either [14]. People simply suppose the current technology enables them to effortlessly collaborate over distance [32].

2.1. Advantages?

The most frequently cited advantage of GSD is that of the *reduced development costs* [30, 14]. By moving parts of the development to lower waged countries, the same work could be done with a smaller investment. While the obtained savings seem like a large benefit at first, there are reasons to believe that GSD introduces complexities that corrode these initial cost savings [14]. Enlarged impediments in communication and structural congruence[3] lead to increased complexity in control, coordination, and elongated response times [14, 26, 11, 22, 21] moderating at least the development speed: conveying what to build and how to work together is considered very difficult [24].

This last consequence mitigates another assumed benefit: *reduced development duration* induced by leverage of time-zone effectiveness. Because the cost of initiating contact is much higher when situated at different locations, developers do not try to communicate as frequently as they would have in collocated environments [21, 16]. Furthermore, the more time-zones crossed, the less time there is that people are at work the same time. This leads to increased tension as all synchronous communication should take place in a smaller time frame [24, 32]. But most

[3] *Structural congruence* refers to the match between the organisational structure and the ability of that organisation to carry out a task. The organisational structure is influenced, among several factors, by the interdependencies amongst tasks. As reseach by Cataldo *et al.* shows, structural congruence is associated with shorter development times [11].

important for this mythical benefit: cycle times increase, even if messages are answered promptly [21]. Hence, different companies tend to shift their working hours in order to maximise overlap, providing more time for synchronous communication [14, 8]. However, it seems that software *quality* does not have to be affected by geographical separation of teams [32], if only the developers take their own responsibility of staying in close contact with other sites [4].

Yet another benefit of GSD might be the possibility to *subdivide the development work into modules* that can be developed in parallel across multiple sites. While the modularity approach has proven to be very useful [36], the relationship between product structure and task structure is not as simple as previously thought [9, 11, 15]. Unfortunately, this model seems to be far from reality [1] and references therein]: although there are some advantages, companies need to be wary of the reduced communication between sites, leading to the persistence of incorrect or conflicting assumptions made, which in its turn can result in problems at the integration stage [14, 26, 11, 31, 21, 16]. The work should therefore be divided into loosely coupled modules: long distance dependencies have to be straightforward and unambiguous [32].

Fourth, the *access to a large pool of skilled developers* would be eased when the company moves to places with large numbers of experienced developers. Different backgrounds could lead to increased innovation and sharing of best practices. Unfortunately, serious cultural differences causing misunderstandings attrite this benefit: because developers have little opportunity to communicate in an informal *ad hoc* manner, the building of mutual trust is impaired. Hence there is little incentive for sharing [41, 14, 26, 38, 23, 32, 21]. People simply are scared to share one's best ideas if they were then to be seen as common instead of unique [32]. Furthermore, they fear loss of control and jobs when work is outsourced [24, 23, 32].

Last, there is the benefit of a *closer proximity to market and customer*. Easier ways to communicate directly with customers could ameliorate the interaction needed in requirements engineering and subsequent software development. Having employees located proximate to the market implies that they are closer to the customer on cultural and linguistical grounds [14, 7]. Unfortunately this comes at an expectable expense: having developers with different cultural backgrounds will introduce the same socio-cultural problems as aforementioned.

2.1.1. Conclusion

The previous information implies that the benefits of GSD cannot be exploited without effective mechanisms for information- and knowledge-sharing [41, 25, 23]: distance still matters [32]. Research indicates that GSD imposes several problems: increased communication delay and work completion time, both due to a lack of informal communication and difficulties in finding expertise and knowledge [8] and references therein]. Therefore, the main challenges lie in the complexity of maintaining good communication and coordination when teams are dispersed [26, 25]. Luckily, there are some ways to lessen the introduced problems.

2.2. Communication, coordination and control

The recurring theme in the previous section clearly is communication and the effects of a lack thereof. Confusions and misunderstandings happen all the time, even with collocated development [1], but they are exacerbated in GSD because the availability of communication channels is restricted to less rich forms. Also, the common ground[4] between the separately located developers is less well established [32]. Moreover, geographic differentiation leads to compartmentalisation[5], increasing in-group cohesiveness, ethnocentrism and unwillingness to trade information, inhibiting cross-site collaboration [40].

The different work groups and companies have different application knowledge and different technical vocabularies that needs alignment [16]. People construct common ground from whatever cues they have: the fewer cues one has, the more likely misinterpretations will occur [32]. For a project to be successful, common representational conventions and terminology need to be established to facilitate effective communication [26, 16]. However, even with all emerging communication technologies, face-to-face contact is essential to both overcome and avoid cultural misunderstandings and enhance the development of trust [24]: distance still affects how humans interact [32].

Complications in communication with GSD have different appearances and dimensions, such as a lack of effective documentation, a lack of synchronisation resulting in incompatibilities when integrating work, cultural differences leading to chronic misunderstanding, the difficulty of sharing knowledge, and a lack of teamness, resulting in trust issues, conflicting work styles and a worse likelihood of helping each other out when necessary [26, 20, 25, 22, 38, 23, 31, 21, 27]. The ability to communicate directly, quickly and effectively is greatly impaired with dispersed software development [21]. However, while effective communication could still take place, the effort to do so is usually quite large [32]. The question remains whether it is possible

[4] *Common ground* refers to the knowledge that different people have in common and that they are aware of the fact they do. Common ground is easily established when collocated, because developers then not only share cultural and linguistic backgrounds, but there is also better awareness about the microcontext – what one's doing, what remains to be done et cetera [32]. Common ground is also known as *transactive memory*.

[5] *Compartmentalisation* is the limiting of information to persons who require it in order to perform their tasks [42].

to accomplish the same engagement of trust and teamness at a distance as when collocated [32].

Communication (and subsequently coordination as well) is of critical importance as it has considerable influence on the success of a development project [26, 20, 21, 27]: it provides the only possibility that separate task groups will be able to consolidate their efforts into a unified system [15]. While developers spend a large proportion of their time communicating [37], a physical separation of a mere 50 meters would result in the end of all regular communication [22]. Suggested practices tend to minimise the perceived distance between the desks in all dimensions: geographical, temporal and cultural [7, 30, 20]. However, the implementation of these practices repeatedly conflicts with cost saving strategies [30], therefore being economised often [26].

2.2.1. Communication tools

One of the ways used to create an environment that puts people in virtual proximity is by the utilisation of tools. Many of the tools used for collocated development – like version control and change management – lend themselves quite well to GSD [20], and the ongoing technological progress has come up with several communication tools to alleviate the remaining problems at minimal cost [38]: improvements in tools allow groups from different locations and backgrounds to come together as a team [39]. To compensate the lack of informal and unplanned "coffee machine talk" people would engage into when collocated [37], new channels for communication were introduced, such as email and instant messaging [10, 38, 31]. Yet, developers still prefer face-to-face communication: using electronic ways to communicate often results in misinterpreted meaning of the question, and a different level of detail in the answer than the recipient required [28, 24, 8]. Furthermore, what appeared to be merely "casual conversations around the water cooler" often served as an important way to exchange knowledge and information critical to project coordination [23, 21]. Research by Batra and Herbsleb showed the significance of this type of communication: their participants were not aware of any need to communicate, leaving the tools introduced for communication inadequate [2, 21].

As awareness about both the status of development at other sites and the knowing who to contact is important for the feeling of 'teamness' [4, 22, 21] and synchronisation amongst sites [10], keeping in contact on a close and frequent basis is vital [26, 37, 16]. Simple tools like an

audio connection and a shared text editor have proven to be insufficient for the job when there is little common ground; people who have established a lot of common ground can communicate well, even over such impoverished media [26, 32]. There have been some attempts to recreate the sense of collocated awareness remotely, but several studies indicate that this has not been fully successful yet: distance still negatively affects communication [2, 8, 32] and references therein]. Even in GSD it still is the human element that is critical and dominant; not the tools. Perhaps we need to look at organisational issues for ways to improve the development process as well [40, 37, 16].

2.2.2. Improved alignment of technology and processes

It is not solely communication that needs improvement: the distribution of software development tasks amongst separated development sites is likely to introduce technical incongruities as well [30, 4, 39, 31, 21]. Sites often differ on issues such as development environments, change and version management systems, development methodology and corporate culture [26, 20, 23, 31]: for example, developers' tool use frequently correlates with their preference instead of the standard of the organisation [28]. Differences like these lead to a wide variety of problems such as misunderstandings, management problems and a lack of awareness about the current status of a project [4, 20], but the greatest problem of all will arise when entering the integration stage [14, 26, 21], as different work groups might have used incompatible development methodologies. Processes have proven to evolve eventually, and because the dispersion of development teams the likelihood that this evolution will develop in a corresponding manner across sites is very low [21]: shared change management systems, the preservation of one code branch and frequent builds can help to overcome these challenges [24].

2.2.3. Reduction of interdependence

Studies have shown that there is an even greater need to communicate when developers are located separately, causing coordination overhead [3]. However, there is a cost increase to communicate effectively in GSD [21, 16]. Also, the way and intensity developers apply communication tools and recommended development methods differs per person and organisation [9, 28, 22]. Because the conflicting nature of those facts, it might be an idea to look for ways to reduce the need for inter-site communication [8, 21, 16].

An ideal arrangement would let the sites operate as independent as possible while providing for easy, flexible, and effective communication across sites [23]. Previous studies have shown that people will automatically reorganise work in such a way that they do not have to rely on tight collaboration with a remote team member, even when this initially was the case [32]. Instead of integrating the work of all development sites closely, the software development task could be divided into nearly independent parts [32], diminishing the needs for communication amongst work groups [28, 21, 16, 15]. By this, one of the advantages of geographic dispersion – a reduction in dependencies amongst modules resulting in easier change management and maintenance – is applied to its full extend [36]. This modularisation approach has proven to be very useful for dividing complex tasks into manageable units [9, 36], but unfortunately it is still vulnerable to an imperfect foresight and the emergence of unexpected events [1, 35, 22, 21].

Software is constantly subject to change [6]. Many people make the mistake to think that extensive and long documentation will pave a clear way for the developers, but this is both not possible and unwanted [5, 16]. Over-engineered documentation only increases interference when events were wrongly anticipated [21]; only in ideal circumstances no further events would occur that affect the nature of requirements after conclusion of the elicitation phase[6] [13]. Informal communication channels are critical to complement explicit coordination mechanisms as way to both resolve unclear details in a matter of minutes and create a feeling of trust [41, 19, 21, 27]: most such interactions are less than five minutes long apiece but have a total of 75 minutes each day [37].

The reduction of interdependencies amongst work groups is a noble goal to strive for; unfortunately, unpredictability of software development projects spoils the game: formal means of communication often cannot keep up with the changes of requirements, increasing the needs for informal communication to solve uncertainties and transfer knowledge [12].

[6] But that is not completely true either: all *successful* software gets changed [6].

2.2.4. Conclusion

Software entities are far more complex for their size than perhaps any other human construct: no two parts are alike [6]. Because there tend to be more people involved in distributed work compared to same-site work, there is an increase in coordination difficulty and an unavoidable need to guide the communication pathways [22]. Greater project sizes come with enlarged difficulty of communication [22, 6].

The combination of communication tools and proper modularisation could ease the communication problems encountered in GSD [23]. Unfortunately, that is not all of it: people still have to be guided to use the equipment in the right way, and to its full extend [9]. Development methods and practices help us to tackle these challenges.

2.3. Development means & methods

Perhaps the need to communicate in an informal manner needs to be acknowledged and other ways than extensive documenting have to be examined. Rather than devising methods to minimise communication, a goal should be to improve efficiency interpersonal communication [27].

Despite the fact that the decrease in practiced communication and increase in needed communication is contradictory [35, 25], the process of iterative and incremental design[7] (IID) suits GSD extremely well [34] since it actually helps to reduce the problems caused by distribution. The communication principles of agile have the potential to alleviate the impediments in coordination and communication [35, 25].

2.3.1. Agile practices

IID being a core practice in agile methodologies [34] and references therein], this is where agile methods can jump in: agile methods promote frequent communication, continuous integration and regular builds [35]. Agile methods are characterised by short, iterative cycles of development driven by product features, periods of reflection and introspection, collaborative

[7] As recognised by Brooks in 1987, systems should not be build but *grown* by incremental development: it should first be made to run with dummy functionality and then be fleshed out. Morale effects are startling when a system first runs, even when it does not do anything. Brooks: "I find that teams can *grow* much more complex entities in four months that they can *build*" [6]. IID is the more developed and expanded version of this idea.

decision making, incorporation of rapid feedback and continuous integration of code changes into the system under development [25] and references therein]. This leaves several benefits: an increase in transparency, instant feedback possibilities, flexibility for customers and developers, and avoidance of 'big bang' integration [41, 18, 34].

2.3.2. Documentation in GSD

In general, developers avoid using design documents when possible, preferring code explorations and face-to-face talks with their team mates to attain understanding of the code [28, 23]. Unfortunately, the simplicity of strolling to another team broke down into difficulty with the introduction of GSD; in GSD developers have to document more meticulous: context and implicit knowledge differs across sites because a less well-founded common ground.

Flexibility and agility are necessary for successful software development, especially in small to medium sized companies – be it distributed or not [2, 5, 1, 15]. However, documentation is a very impoverished and slow-paced form of communication [24]. The nature of software requirements and design is to be unstable; documents in general are not, therefore easily getting out-of-date [26, 9, 12, 22, 16]. The focus has been for decades on developing structure in documentation, while the true reasons for which they have been developed were slowly forgotten [1]. We have to avoid a means-end inversion where documents are made because we have to, instead of the additional value they were intended to bring [1]. By avoiding to over-engineer a process description and trusting on the creativity of developers, we can reduce interference with documentation that anticipated on future events wrongly [21, 25] and references therein]. The question remains: is there actually even any need documents itself?

The lack of rich communication leaves story cards insufficient because one cannot rely on tacit knowledge alone; there is a need for supplementary documentation [41, 2, 1]. At all times, the design should stay simple, using low-tech techniques to create a shared group visualisation of the project progress [25]. This allows for an improved awareness and visibility of the ongoing process, leading to more mutual trust and an increased feeling of responsibility [25, 35]. Documentation is still important, but it has to be concise and, even more essential, congruous with the current state of the project [1, 23].

2.3.3. Communication with agile methods in GSD

Agile methods are basically an attempt to satisfy the quest for more lightweight and faster development [25]. Since designs never exhibit perfect modularity and are never error-free, process executions rarely flawless, and the world is never completely predictable, informal communication will be essential to maintain project coordination [21]. Hence, informal communication should be encouraged [2, 34].

Despite this, formal communication is a necessary complement to the informal channels [23, 27]. Elementary probability theory tells us that the number of possible communication paths is approximately half the square of the number of people in an organisation, which in turn tells us that sheer informal communication will soon become too burdensome [15]. Formal communication could be realised through for example (virtual) meetings and deployment of liaisons[8]. Meetings allow the diverse groups involved to come together in a regular, controlled way [27]. Liaisons should be in an *agile team*, together with some programmers and designers. The agile team's purpose is to communicate on a daily basis in order to evaluate and monitor the project. This group would then be a hub facilitating communication and coordination [2].

2.3.4. Conclusion

It is clear that frequent communication is a central prerequisite with the application of GSD in general [8], and especially for successful implementation of IID in GSD [34]. Agile methods bring ways to alleviate the problems encountered with dispersed development, but there are principles that just are not feasible when deploying agile methods in a GSD context [2]. It should be noted that, although most studies report that agile development practices are easily adopted and work well, it seems that the principles work best – or only – with relatively small teams [2, 18, 5], but further research is needed here.

Some kind of hybrid agile and document-driven development appears to fit GSD best, as it harmonises agility and discipline [2]. The combination of concise documents and the encouraged use of formal and informal communication is a very strong one, as it combines the best and leaves the most impracticable, unfeasible principles of both worlds behind [2, 18, 1].

[8] Typically, a *liaison* is someone who has spend a reasonable period of time in the country whereto the offshoring takes place. Its informal role is to facilitate the cultural, linguistic, and organisational flow of communication [8].

3. The future of Global Software Development

3.1. Introduction

It is unlikely that the advent of GSD will come to an end. The additional impediments in communication and coordination with GSD require solutions in order to remain or become profitable. There exist three approaches to tackle the problems and they need to be applied all: the technical- and documentation-routes, and the development of teamness. All provide a means to enhance cross-site coordination.

3.2. Improvement of coordination and awareness

3.2.1. Tooling and communication technologies

New tooling could reduce the perceived distance between physically separated development teams. Because people tend to adapt their behaviour rather than fix the technologies used, emphasis should be put on development in a user-centred manner [32, 37]. Despite this, new tools should always support current practices first before trying to enhance them: people have to get convinced of its use. Adoption simply takes time: innovations should be introduced in small steps [32, 17].

First, to increase the ease of communication, new tools need to facilitate the use of multiple channels together (i.e. multiple applications combined with audio and video stream) to approach the normal face-to-face way of work [8]. Integration of currently existing IDEs with communication tools could be such a solution, as developers than have to switch less between different tools and environments [28, 24, 6]. An IDE must become a medium of communication to integrate people, tools and information [16].

Second, increased awareness and ease to stay up-to-date should be achieved: IDEs should not merely be integrated with, for example, version control management and new communication channels: it should make it easier to find information, check availability of colleagues and remain aware of changes in software related to the developer's activities. In addition, more effective cross-site meetings, both planned and spontaneous, should be facilitated [21]. Because the ease of getting information is a critical determinant in choosing what source to draw upon,

tools should enable developers to exchange knowledge and help them finding the right expertise [27]. Tools should help others see where the most recent activity took place to enable developers in avoiding interference with ongoing work, and identify conflicts in between requirements [17, 16, 27]. Furthermore, tools must accommodate changes in the software design as an ordinary process and support the representation of uncertain design decisions [16]. Third, keeping documentation up-to-date should be an easy task, only then developers would use the design documents as reference, and not the code [28, 23].

In even further future, there might be technical ways to not only replace face-to-face communication, but even augment it. However, Olson and Olson remark that they feel that several key elements of interactivity will be very resistant to support [32]. For example, primitive Virtual Reality is already available in software development, but the use of this technology in this domain is only scarcely out of its egg, yet [32].

Concluding from the above, new tooling should introduce a virtual proximity in which developers can easily communicate on frequent basis. Identification of conflicts in requirements and executed work should be done by tools instead of humans.

3.2.2. Documentation

Second, there is documentation as a form of communication. Proper modularisation reduces dependencies across sites and hence the need for communication. However, as development is rarely a fully predictable undertaking, it should accommodate easy changes and means to make those changes noticed by the developers involved as well. Furthermore, better capture of informal, transient designs, and a decrease in necessary effort to keep them up-to-date could do the trick [28]. When documenting, an upper limit of about two pages per artefact is advised: the majority of developers simply do not want to read more [1].

The solution to the problems introduced by GSD is neither to add more documentation nor to abandon it: it is to get better documentation. We must go towards lean, concise documentation [1]. The combination of a well established common ground, easily accessible knowledge and concise documentation appears to fit GSD best.

3.2.3. Teamness

Apart from the application of new communication technologies, the feeling of teamness is a determinant in the success or failure of a software project as well. *Teamness* could be established by better communication technologies as described in section 3.2.1, but the establishment of common ground should be facilitated in other ways as well.

Common representational conventions need to be established for successful development. Research has revealed that, for example, cultural issues could be overcome fairly quickly: if only one visits the other site on a relatively frequent basis [24]. The more common ground one has, the higher the productivity, because more effective answers could be given. Knowing the context and knowledge of the other involved people increases effectivity [26, 12, 32]. Establishing common ground could be done by sending liaisons [24].

Second, the maintenance of an extensive interpersonal network is associated with better project outcomes: the constructed greater awareness aids inter group coordination [27]. Research by Shami *et al.* showed that developers without a collaborative history tend to form strong in-group biases that inhibit cross-site collaboration: compartmentalisation [40]. Using tools and liaisons, both the allocation of appropriate knowledge should be eased and the willingness to share should be increased.

In conclusion, the maintenance of long-term relationships and a well established common ground between the different participating development sites alleviates the problems thwarting a feeling of teamness.

3.3. Conclusion

To improve processes and design, first a concise, accurate and meaningful information about the existing situation must be obtained [37] and references therein]: what are the exact problems and their possible solutions. Better tooling, improved teamness and more appropriate documentation should be applied all together to reduce the communication difficulties. Unfortunately, research by Smite *et al.* concluded that – up till now – there exists still no recipe for successful and efficient performance in globally distributed software engineering [30].

4. Implications for the SAD

4.1. Introduction

In GSD, the communication problems accompanying collaborated work are exacerbated. As concluded in section 3.3, the solution is neither to add more documentation nor to abandon it: it is to get better documentation. We must go towards lean, concise documentation [1]. Regardless of this, there is reason and evidence to put both more and less emphasis on documentation in GSD environments.

4.2. Contra documentation

Many developers see documentation as a weak form of communication, pointing at the tardiness, incompleteness and ambiguities [16]. As a result, developers prefer exploration of the code to find answers. When that fails, they still favour the consulting of knowledgeable team mates rather than design documents [28]. In addition, all forms of written documentation are judged as less valuable than personal contacts [27]. Other properties, like visualisations in documentation, are rarely used for the core of the development process [12].
There exist plenty of reasons for a developer not to record changes: the effort of checking out the code, editing it and checking back in, possibly triggering all sorts of conflicts, is enough to dissuade him. This leads to the fact that lots of design information is kept in peoples' heads and never written down [28, 23].

4.3. Pro documentation

The aforementioned reasons are all directed towards tardiness: it is too difficult to keep documentation up-to-date. As was already assessed, future tools should accommodate easy changeability. Also, they should be able to actively notify the developer of any changes. Only if documents are always up-to-date and easy to find, they could become a way of communication [28, 13].

4.4. Conclusion

Documentation as a means of formal communication is important: it eases the coordination and improves modularisation of work to a great extend. Of course, informal mechanisms of communication to complement documentation are necessary as well, but the required communication could be channelled by documentation in order to keep it manageable.
Accurate and up-to-date, straightforward to understand and easily updated documentation is necessary in larger software projects, and especially in GSD.
Unfortunately, people are reserved when it comes to new techniques: when they have experience with tardy and ambiguous documentation, they will not adopt the new and improved type without ado: once burned, twice shy [32]. Introducing new technologies and methods takes time.

5. Conclusion

Software development is not an isolated activity: no single developer has the ability to create or even fully understand large complex systems [11, 28, 27, 15]. Hence, most developers spend a significant part of their day communicating with various other co-workers [37]. Unfortunately, collocated development becomes increasingly impracticable as the needs for quick and cheap development are rising, therefore pushing companies to Global Software Development [39]. However, this aggravates most possibilities for unplanned, informal communication [2, 16], leaving devastation behind: hindered communication will pave the way for misunderstandings, frustration, and ultimately failure of a successful ending of a software development project [26, 16]. The need to communicate with separately located co-workers should be lessened by proper modularisation in which the modules are loosely coupled [28, 23, 32, 36, 15]. The remaining required communication should be supported by a well established common ground and good inter site relationships. Communication tooling, liaisons, and training could provide help in founding such a common ground, alleviating most of the communication problems encountered [26, 24]. However, documentation is important in outsourced projects too: one cannot rely on tacit knowledge alone as there is less *group knowledge*, especially when the collaboration is for a short period of time [2, 16].

In spite of all this, distance will continue to matter [32] and collocated development remains the best option regarding software quality and cycle time. All we can do is try to approach the experience of face-to-face communication as close as possible.

6. References

[1] AGERFALK, P. J., AND FITZGERALD, B. Flexible and distributed software processes: Old petunias in new bowls? *Communications of the ACM 49*, 10 (October 2006), 27–34.

[2] BATRA, D. Modified agile practices for outsourced software projects. *Communications of the ACM 52*, 9 (September 2009), 143–148.

[3] BIANCHI, A., CAIVANO, D., LANUBILE, F., RAGO, F., AND VISAGGIO, G. Distributed and colocated projects: a comparison. In *Proceedings of the seventh workshop on emperical studies of software maintenance* (November 2001), pp. 65–69.

[4] BIRD, C., NAGAPPAN, N., DEVANBU, P., GALL, H., AND MURPHY, B. Does distributed development affect software quality? an empirical case study of windows vista. *Communications of the ACM 52*, 8 (August 2009), 85–93.

[5] BODEN, A., NETT, B., AND WULF, V. Coordination practices in distributed software development of small enterprises. In *International Conference on Global Software Engineering* (2007), pp. 235–244.

[6] BROOKS, F. P. No silver bullet: Essence and accidents of software engineering. *IEEE Computer 20*, 4 (1987), 10–19.

[7] CARMEL, E., AND ABBOTT, P. Why 'nearshore' means that distance matters. *Communications of the ACM 50*, 10 (October 2007), 40–46.

[8] CARMEL, E., AND AGARWAL, R. Tactical approaches for alleviating distance in global software development. *IEEE Software 18*, 2 (March/April 2001), 22–29.

[9] CATALDO, M., BASS, M., HERBSLEB, J. D., AND BAS, L. On coordination mechanisms in global software development. In *International Conference on Global Software Engineering* (2007), pp. 71–80.

[10] CATALDO, M., AND HERBSLEB, J. D. Communication networks in geographically distributed software development. In *Proceedings of the ACM 2008 Conference* (2008), pp. 579–588.

[11] CATALDO, M., WAGSTROM, P. A., HERBSLEB, J. D., AND CARLEY, K. M. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Computer Supported Cooperative Work: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (November 2006), pp. 353–362.

[12] CHERUBINI, M., VENOLIA, G., DELINE, R., AND KO, A. J. Let's go to the whiteboard: how and why software developers use drawings. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (April-May 2007), pp. 557–566.

[13] CHISAN, J., AND DAMIAN, D. E. Towards a model of awareness support of software development. In *The 3rd International Workshop on Global Software Engineering* (May 2004), pp. 28–33.

[14] CONCHÚIR, E., AGERFALK, P. J., OLSSON, H. H., AND FITZGERALD, B. Global software development: Where are the benefits? *Communications of the ACM 52*, 8 (August 2009), 127–131.

[15] CONWAY, M. E. How do committees invent? *Datamation 14*, 1 (April 1968), 28–31.

[16] CURTIS, B., KRASNER, H., AND ISCOE, N. A field study of the software design process for large systems. *Communications of the ACM 31*, 11 (November 1988), 1268–1287.

[17] DEKEL, U. Supporting distributed software design meetings: what can we learn from co-located meetings? In *Proceedings of the 2005 workshop on Human and social factors of software engineering* (2005), pp. 1–17.

[18] DYBA, T., AND DINGSOYR, T. Empirical studies of agile software development: A systematic review. *Information and Software Technology 50* (2008), 833–859.

[19] HARGREAVES, E. J., AND DAMIAN, D. E. Can global software teams learn from military teamwork models? In *The 3rd International Workshop on Global Software Engineering* (May 2004), pp. 21–23.

[20] HERBSLEB, J. D. Global software engineering: The future of socio-technical coordination. In *FOSE'07* (2007).

[21] HERBSLEB, J. D., AND GRINTER, R. E. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st International Conference on Software Engineering* (1999), pp. 85–95.

[22] HERBSLEB, J. D., AND MOCKUS, A. An empirical study of speed and communication in globally distributed software development. *IEEE transactions on Software Engineering 29*, 6 (June 2003), 481–494.

[23] HERBSLEB, J. D., AND MOITRA, D. Global software development. *IEEE Software 18*, 2 (March/April 2001), 16–20.

[24] HERBSLEB, J. D., PAULISH, D. J., AND BASS, M. Global software development at siemens: experience from nine projects. In *Proceedings of the 27th international conference on Software engineering* (May 2005), pp. 524–533.

[25] HOLSTRÖM, H., FITZGERALD, B., AGERFALK, P. J., AND CONCHÚIR, E. . Agile practices reduce distance in global software development. *Information Systems Management 23*, 3 (June 2006), 7–18.

[26] KOTLARSKY, J., VAN FENEMA, P. C., AND WILLCOCKS, L. P. Developing a knowledge-based perspective on coordination: the case of global software projects. *Information and Management 45* (2008), 96–108.

[27] KRAUT, R. E., AND STREETER, L. A. Coordination in software development. *Communications of the ACM 38*, 3 (March 1995), 69–81.

[28] LATOZA, T. D., VENOLIA, G., AND DELINE, R. Maintaining mental models: A study of developer work habits. In *International Conference on Software Engineering* (May 2006), pp. 492–501.

[29] LEE, J., HUYNH, M. Q., KWOK, R. C., AND PI, S. It outsource evolution - past, present and future. *Communications of the ACM 46*, 5 (May 2003), 84–89.

[30] ŠMITE, D., WOHLIN, C., GORSCHEK, T., AND FELD, R. Empirical evidence in global software engineering: a systematic review. *Empirical Software Engineering 15* (2010), 91–118.

[31] MOCKUS, A., AND HERBSLEB, J. D. Challenges of global software development. In *Proceedings of the seventh international software metrics symposium* (2001), pp. 1–3.

[32] OLSON, G. M., AND OLSON, J. S. Distance matters. *Human-Computer Interaction 15* (2000), 139–178.

[33] O'ROURKE, K. H., AND WILLIAMSON, J. G. When did globalisation begin. *European Review of Economic History, Cambridge University Press 6*, 1 (April 2002), 23–50.

[34] PAASIVAARA, M., AND LASSENIUS, C. Using iterative and incremental processes in global software development. In *The 3rd International Workshop on Global Software Engineering* (May 2004), pp. 42–47.

[35] PAASIVAARA, M., AND LASSENIUS, C. Could global software development benefit from agile methods? In *IEEE International Conference on Global Software Engineering* (2006).

[36] PARNAS, D. L. On the criteria to be used in decomposing systems into modules. *Communications of the ACM 15*, 12 (December 1972), 1053–1058.

[37] PERRY, D. E., STAUDENMAYER, N. A., AND VOTTA, L. G. People, organizations and process improvement. *IEEE 11*, 11 (July/August 1994), 36–45.

[38] PRIKLADNICKI, R., AUDY, J. L. N., AND EVARISTO, R. Global software development in practice. lessons learned. *Software Process: Improvement and Practice 8* (2003), 267–281.

[39] PRIKLADNICKI, R., AUDY, J. L. N., AND EVARISTO, R. An empirical study on global software development: Offshore insourcing of it projects. In *The 3rd International Workshop on Global Software Engineering* (May 2004), pp. 53–58.

[40] SHAMI, N. S., BOS, N., WRIGHT, Z., HOCH, S., KUAN, K. Y., OLSON, J., AND OLSON, G. An experimental simulation of multi-site software development. In *Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research* (2004), pp. 255–266.

[41] SHRIVASTAVA, S. V., AND DATE, H. Distributed agile software development: a review. *Journal of Computer Science and Engineering 1*, 1 (May 2010), 10–17.

[42] WIKIPEDIA. Compartmentalization, http://en.wikipedia.org/wiki/Compartmentalization_(intelligence), July 2010.

[43] WOLF, T., NGUYEN, T., AND DAMIAN, D. E. Does distance still matter? *Software Process: Improvement and Practice 13* (2008), 493–510.

Appendix B

The Analysis

transcribing, encoding and theorising

Overview

This appendix provides the reader with an introduction on the methods used in the analysis phase and give rationale for the decisions made to get well-founded results. In general, each section will start off with a description of the theoretical background behind the property, process or method, then proceed with a section about the merits and demerits, and conclude on how we used it in our experiment.

---

1. Introduction

After gathering data it needs to analysed in order to be able to derive theories from it. The collected data in our experiment consists of video and audio fragments of subjects looking for information that could help them answering the question we asked them. In our experiment, we chose audio as the most important source of data, as we asked the subjects to think aloud while working on the questions.

The founding of new theories could be done without transcribing or encoding, which allows access to prosodic and paralinguistic features of the data. Unfortunately, direct analysis of data cannot give us a good *overview* of what we are actually doing. To keep grip on the massive amount of data produced by this experiment, we need to transcribe and encode the recorded audio.

Although it is usually the case that in the process of encoding only more data is generated, it is this additional data that permits us to find general patterns and concepts, which is exactly the goal of our experiment: the verbal protocols produced in our experiment are qualitative data from which we want to infer meaning and draw inferences about what the participants think [9].

2. Methodologies

2.1 Protocol Analysis

In Protocol Analysis, the processes of solving a problem or making decisions are the focus. The processes of problem solving correspond to the sequence of problem states the subject undertakes as he applies permissible operators [4].

Retrieving information about the cognitive processes occurs through a method called Think out Loud/Talk out Loud (ToL). Several studies show that the thoughts verbalised correspond to the usually covert thought activity. Therefore, they are useful to obtain the strategies subjects use to get to an answer without affecting the ongoing cognitive processes during problem solving [5].

While it may sound as a controversial method of gathering data and performing analysis, it is not that alien: in actual life we frequently find ourselves explaining something to others or trying to communicate our thinking. In addition, when a task is difficult or an intricate text has to be read, we tend to vocalise our thoughts – especially in noisy environments. The corresponding elements of these examples show that in most cases we are not even aware of the fact we vocalise; in certain situations, people even find it almost impossible to suppress these overt verbalisations. This evidence strongly implies the absence of overhead any vocalisation might cost us[1]. Moreover, what subjects report during a ToL session, most often is what the researcher wants to hear. All the aforementioned greatly supports both the relevance and nonexistent impact of thinking out loud on problem solving [5].

[1] On a level 2 at most. There exist 3 levels of thinking out loud [5: 79]:

1. Mere thoughts;
2. Explications of thoughts (decreases speed of cognition but leaves these processes undisturbed) ;
3. Explanation of thoughts (requires links to the LTM and therefore decreases reliability of verbalisations).

2.2 Grounded Theory

2.2.1 What is Grounded Theory

Grounded Theory (GT) is an inductive form of qualitative research where data collection and analysis are conducted together. With GT one does not establish hypotheses prior to performing research; the researcher *creates* one after profound analysis of all data available. The main claim in GT is that theories remain grounded in the observations rather than being generated in the abstract [6, 9].

2.2.2 Goals

The goal of GT is to offer the reader a conceptual explanation of a latent pattern of behaviour that holds significance within the social setting under study. It is the abstract concept that is under study, not the descriptive detail [1: 268]. The skill of a grounded theorist is to abstract concepts by leaving the detail of the data behind, lifting the concepts above the data and integrating them into a theory that must explain, not merely describe what is happening [1: 272-273]. GT should eventually come up with a coherent account of the phenomena under investigation, clustered around one or two core categories derived from the data [9]. Bottom line: the essential quality of GT is that it makes sense when applied properly: its outcome should be relatable to actual situations [1: 114].

2.2.3 Principles and concepts

GT consists of two basic principles of category[2] building:

1 Categories should emerge from the data; they should not be forced onto it;

2 Researchers should be able to see relevant data and reflect upon it using theoretical terms. This is called *theoretical sensitivity*.

Unfortunately, by the contradicting nature of those two principles they are difficult to reconcile. Hence, one needs to see the "emergence of categories out of data" in an epistemologically informed way: the development of categories is dependent on the availability of theoretical concepts [1: 193 & 206].

[2] Categorising is the analytic step in GT. The researcher selects codes that are applied to transcripts and tries to combine them into a more abstract, theoretical code: a category. More information about categorising can be found in section 6.

GT benefits from low falsifiable or *sensitising* concepts (in contrast with quantitative research with which high falsifiable concepts are preferred from initiation onwards). Sensitising concepts do not force existing categories or preconceptions onto data; instead they serve as a heuristic device, merely showing a direction along which to look for answers, hence helping to identify theoretically relevant phenomena [1: 207]. However, in order to generate useful theories, eventually – when reached fairly advanced stages of the theory building process [1: 212] – high falsifiable hypotheses should be formed [10].

GT requires that the researcher enters the research field without any preconception of problem statements or literature reviews in order to remain truly open to exploring the area and let a new theory emerge out of the data obtained [1: 269]: in an ideal case, the search for literature related to the experiment comes *after* the construction of the grounded theory [1: 123]. However, some researchers counter this principle by mentioning that at least some notion of the field under investigation could alert them for gaps in theorising and help them see whether something discovered is interesting, useful or even undiscovered before [1: 254]. Also, GT is regarded difficult, especially for beginning researchers. In our research we will not stick to the strict GT principle described here. Rather, in order to get a direction in which to search for interesting phenomena or how to explain situations and emerging correlations the alternative stance will be pursued: literature on multimedia principles, cognitive principles and guidelines on communication will be explored during the analysis stage.

2.2.4 Why GT

Because we do not want to investigate hypotheses generated beforehand – we simply have no founded clue of how people would handle the proposed problems and questions – an alternative research method than one from the old-fashioned hypothetico-deductive corner should be applied. Luckily, GT provides exactly the tools to do that: it is particularly useful in situations where little is known about the topic or when a new approach to a familiar area is required [9].

2.3 How to combine Protocol Analysis and Grounded Theory

2.3.1 Differences and similarities

There exists one fundamental difference between GT and PA: where with PA one needs to identify the coding vocabulary in advance, GT almost screams for open coding without any preconception about the research domain. Regardless, both methods unsurprisingly are proponents of objective theorising. PA wants to achieve this by extensive task analysis beforehand to predetermine some of the operations and goals a subject could have during the experiment [5: 276]; GT wants to approach objectivity by only looking at the data and deriving every code and category from appearing actions in order to prevent a theoretical framework from being forced onto the data.

2.3.2 Our implementation

We will combine the two methodologies by abandoning the requirement of a predefined coding scheme as declared in PA to remain open and as unbiased as possible towards discoveries of new patterns or unexpected processes. We only keep the rules as that are to be followed while conducting ToL "interviews" and ignore the rest [5]. In fact, the method we applied is most strongly related to GT: the only divergence is that we already finished the data-gathering phase before starting with the analysis. However, we still generate some kind of a new theory based on the observations made, and our hypothesis will be constantly adapting to new data added during analysis, following all the principles of GT. Therefore we think it is legitimate to build onto this theoretical framework proposed by Glaser and Strauss in their first methodological book *The Discovery of Grounded Theory*.

In addition, after having developed a new theory or hypothesis we could still apply such task analysis to see if there are any similarities between the two. Any occurrence of corroboration between the two will only increase its suspected correctness and our confidence in the emergent theory[3].

[3] Strengthening of theories is possible by the use of the Bayesian Approach. The result of a task analysis could be seen as evidence for the developed theory. When this evidence supports the theory developed, the probability of its correctness is increased [2]. To determine which theory is best, Bayes' Theorem could be used [5: 281].

2.3.3 Conclusion

In the end, it seems only abundantly clear to combine the two methods: in PA one stimulates the subject to talk about whatever they think, do and look at during some kind of task. We could safely presume we do not know anything about the actual cognitive processes going on in the heads of our particular subjects. Therefore they are black boxes to us: we could have *ideas* about the underlying processes, but we will never *know* unless they tell us. The verbal protocols this process of verbalisation produces contains unbiased data from our subjects about their thoughts. Therefore we assert that data gathered using ToL is pre-eminently suitable for the construction of a grounded theory.

3. Doing Grounded Theory

After having defined the conceptual framework we chose to use, we will have to be more specific about how to concretise this. This section will describe every step we have to go through to get to an intelligible intermediate result, ready to be analysed further into a concrete theory or hypothesis.

The practice of encoding leaves several questions to be answered:

1. What to transcribe;
2. How to segment the transcripts;
3. What to code;
4. How to design a coding scheme;
5. How to categorise.

The plain demarcation between the items in the abovementioned list might not be as clear in practice, but for the sake of simplicity, we deal with them on individual basis here. Section 4 through 7 will try to explain the problems and come up with solutions to them we think are best. Section 8 then ties all ends together and describes how to devise a theory based on all preliminary work.

4. Transcribing

4.1 Choosing samples

Due to the time consuming nature of Protocol Analysis, verbalisation recordings are usually sampled or reduced [7]. In fact, it is even encouraged to purposely select data samples – which is different from selecting samples out of convenience – because it is believed they can contribute to the topic under investigation and make easier identification of variations within the new developing theory possible. If applied well, this process of selection does not impair the rigor of the research but even improves it. However, it is important to define what we are looking for and to define these criteria explicitly [1, 9, 11]. Excellent participants meet the following requirements [1: 231]:

- They are known with the phenomenon under investigation;
- They are willing and have the time to participate;
- They are articulate.

In spite of the properties an excellent participant should have, we need to start our analysis with *convenience sampling* to get an overview of the research domain. After having constructed this outline, we can move on to the next stage of sampling: *purposeful* or *theoretical sampling*. When a potential core variable has been identified, subsequent data collection could be delimited to that which is relevant to the emerging conceptual framework: the process of data collection is controlled by the emerging theory [1: 280]. With purposeful sampling one can try to find participants that confirm or disconfirm what the first stage of sampling brought us to advance the developing theory [1: 117]. The new data should be selected to falsify[4] the hypothesis in order to obtain the widest range of possible occurrences conforming to the premises. In this way, our theory should be able to meticulously describe the actual

[4] Because confirmation or verification is logically impossible, *falsificationism* was introduced in the first half of the 20th century. The aim here is to find data to falsify the current hypothesis in order to strengthen it. When conflicting data is found, there are several solutions to overcome that problem. First, one chould change the hypothesis in a way that it from thereon will account for both the formerly conflicting and expected occurances. Second, the observations itself could be refuted or thirdly, the theory could be renounced and a new hypothesis could be devised. However, we should be wary of ad-hoc changes to the hypotheses: those only add more constraints and will make the theory less applicable and hence weaker [2, 10]. (Unfortunately these are not the only problems with naive falsificationism. However, the general idea still holds for our case. For a broader, yet still very surveyable account on Philosophy of Science in general, please take a look at [2]).

phenomenon. Only in this way it is possible to capture most variation within the research domain, hence allowing us to terminate the sampling when no further saturation[5] is achieved [1: 235-239].

Grounded Theory, along with other types of qualitative research, does not rely on notions of statistical representativeness to make claims about the generalisability and authenticity of the findings. We use the best (or worst) cases as those contain the most clear examples of the phenomena under investigation; only if we have made those phenomena apparent, we could use the average samples as well [1: 234]. Looking for deviant cases enables to ensure no important cases were missed that might lead to question the applicability of the newly generated theory [6: 96]. However, we need to be wary for outliers: if a participant is a *true* outlier (i.e. in both quantitative and qualitative respects) he or she may be ignored [1: 240]. In addition, a sample size of around 20 and 30 interviews or hours of experiment is recognised to deliver enough data to construct a well-founded GT [1: 117].
As should be clear from the few paragraphs above, pure randomization during sampling is not an option in qualitative research. When using random samples, the interesting factors would be distributed normally: this would leave us with lots of data about common events and inadequate data about the less common events. In fact, by random sampling we would even *introduce* a bias: by not attending to the meaningful scope of the phenomenon under investigation we would gather lots of data not useful for saturation of the codes, categories and theory. This clearly is inefficient [1: 234].

Following the theoretical background depicted above, we will define our criteria as follows: we will search for data of subjects who were at least moderately outspoken during the experiment. This will give us the most clear view on the strategies those particular subjects used. Avoiding the relatively silent subjects saves us much time while we assume they could contribute only little to the theory in production. Although the common way of reducing data volume is to process only a sub-set of all subjects' protocols [7], this is based on the assumption that each

[5] *Saturation* is the process in which the forming of a grounded theory has not finished. Every phase in GT has to reach its own saturation before one can move on to the next. Saturation could be explained as the cristallisation of codes, categories or the theory. Saturation is reached when the researcher does not perceives anything new anymore [1: 117]: when saturation is reached, there is no new data that could not be accounted for by the grown hierarchy of codes, categories, and eventually the grounded theory.

subject participates around 1 to 2 hours in an experiment; our experiment will take only about half an hour, so sampling *within* a protocol would only leave us an inadequate amount of data. Unfortunately, due to time constraints, we will not carry on with the stage of purposeful sampling after we have constructed an outline of the phenomena occurring. Furthermore, due to the same time constraints, we will be only able to transcribe, code and segment around 10 participants, resulting in approximately 5 hours of data to be analysed.

4.2 What to transcribe from the samples

Intonation, stress and pauses are not easily transcribed [5]. Luckily, in transcriptions it is neither usual nor necessary to transcribe these prosodic, paralinguistic or extralinguistic[6] elements [9]. However, as these elements might indicate a shift of cognitive structures, – an active mental model as content of the WM – they could prove useful when segmenting the data [5: 221-225]. Therefore, to get data as sensitive as possible, we do transcribe pauses and hesitations, but leave stress and intonation out of the transcriptions.

A thing to be wary of are meta-comments made by the subjects. Meta-comments are a subject's attempts to verbalise its own cognitive regularities [5: 311]. Such comments are of little value as the LTM plays too big a role in this process: the LTM is unreliable, inaccurate and reports from it are almost indefinitely incomplete: meta-comments should therefore be at least taken very cautiously [5: 115-116]. Luckily, determining whether a verbalisation is a meta-comment is relatively easy: in most cases the information in such verbalisations will not be used in subsequent behaviour [5]. However, we will transcribe meta-comments if they occur, and deal with them in the encoding stage when needed.

To increase reliability verbalised terms could be replaced with synonyms, but this may result in a loss of semantic content [5]. To avoid this, we will not replace general terms with synonyms, but only references to one of the two presented documents: this will make it easier for us to search through the transcripts without troublesome tricks.

The capturing of what people are looking at defines what they see: their stimulus [5: 278]. We have video data that contains this information, and because of the valuable information it contains, we will use it in our transcriptions by mentioning in which direction they were looking during particular verbalisations. However, the camera will not capture the exact gaze of

[6] *Paralanguage* refers to the non-verbal elements of communication used to modify meaning and convey emotion. *Prosody* is the rhythm, stress and intonation of speech. *Extralinguistic elements* are mostly gestural means of expression.

the participant and hence we will not have data about what the participants are looking at specifically; such data could be obtained by using an eye-tracker. For more information we refer to other studies [8, 12]. Because of this little woolliness, we have to be a bit loose on the meaning of the rough gaze of a participant into a certain direction – the exact stimulus remains unknown.

5. Segmenting

Before we can commence coding, we have to decide how we want to segment the transcripts. Although there is no standard definition of a segment, there exist a few issues to consider [4]:

1. *Granularity*

   Varying unit sizes could be utilised, such as at proposition, sentence, idea, paragraph or specific event level. A decision about the grain size has to be made a priori: using a finer grain size increases data sensitivity, but at laborious costs.
2. *Correspondence between granularity and the objective of research*

   For one type of research an appropriate unit size might be a reasoning chain, which usually involves a grain size of several sentences, while others would want to gain insight in atomic events of a process. In general, one needs to worry about whether the chosen grain size is appropriate to interpret the results meaningfully.
3. *Characteristics of the data*

   When participants need more sentences to convey an idea, it might be useful to group these sentences together to get meaningful units of information.

Units of articulation correspond to integrated cognitive structures. Different cues like pauses, contours, intonation and syntactical markers can be used to segment encodings, but encoding based solely on pauses might not be sufficient; larger syntactic or semantic units are then required to keep each verbalised segment context independent [5: 205, 225, 279; 7]. Because highly related concepts are usually activated simultaneously, related features often produce a clustering effect in retrieval of cognitive structures from LTM. As a consequence, connected information tends to co-occur: efficient segmenting occurs with unit sizes multiple sentences. In addition to the support of segmenting on semantic features, primary cues for differentiating thoughts are their verbs and tenses [5: 223]. This again indicates that semantic boundaries are most important, and that they could be found in many different ways. Based on the previous, it proves often psychologically more meaningful to use semantic boundaries [4].

Line-by-line segmentation (and hence coding) is recommended by many grounded theorists, because it forces analytic thinking with the researcher whilst staying close to the data. We will come back to this in section 6.1.2 and explain why we will not apply this technique in our case.
The only segmenting applied, is the dividing of complete transcripts into segments containing complete answers, one segment per question. Each segmented unit should be large enough to contain all information for the encoding decision to be made in the next phase, and with segmenting per question, this doubtlessly is the case [5: 290]. Pauses and hesitations are preserved for the use of data interpretation. These noncontent elements enable us to determine from the transcriptions whether a participant was busy with one of the two media in silence or when he was verbalising thoughts.

6. Coding

After segmenting the transcripts into units of more or less context independent data is done, it has to be coded. Coding is the first step in the creative analysis of the data: it is used as a way of both managing and organising data, and for the development of clusters, hierarchies and eventually categories [6: 39, 1: 253]. The aim of coding in general is to define what the data are about [6: 38]. In short, codes form a focus for thinking about the text and its interpretation [6: 40], allowing to know more about the field we study, yet carrying the abstraction of the new [1: 84].

6.1.1 Coding cycles

In general, two distinct cycles of coding could be discerned, each acting at a different level of abstractness. The first coding cycle is fairly simple and direct. During this cycle, low level, descriptive, *in vivo,* and/or structural codes are used to identify pieces of data. The second cycle is more challenging as it requires more analytic skills. During the second coding cycle, data is reorganised and reanalysed: categories and subcategories, core themes and relations amongst them are then developed [11: 45].
Most of the literature about qualitative research distinguishes second cycle coding into a different part called *categorising*. We assert this dissection is fairly grounded on the basis that categorising takes not only place at a whole different level of abstraction, it also involves the profound use of memos in the art of picturing categories and core themes. Therefore the

categorising phase will be discussed in the separate section 7. Hence, the continuation of this section only handles first cycle coding methods.

6.1.2 Line-by-line coding

Segmentation was based largely on semantic features in order to get meaningful units of information which are context independent or require only a very narrow one. However, there is exists a method of coding that could be applied independently of the segmentation, directly onto the initial transcripts. With *line-by-line* coding, the researcher takes each line in the transcripts – which almost never constitutes a complete sentence – and tries to encode it according to its content. This approach has one advantage: it forces the researcher to look closely at the data which minimises the chance of missing out on important information [1: 275]. Although it should merely be a device for the initial breaking down of data – not to track down complete and true meaning [1: 196] – it nevertheless creates much overhead during analysis and its profits are questionable. As nicely put by Stern: "I never do a line-by-line analysis because there is too much filler to skip over. Rather, I do a seizure operation looking for cream in the data" [1: 118]. We will stay with her: we segmented the data according to certain principles and during coding, we will stick to this. So, no line-by-line coding for us.

6.2 What to code

There are several sources of data that could be coded. In GT, these usually consist of not only recorded audio and video, but field notes and memos made by the researcher as well. All of these should be taken into account during the process of encoding.

As mentioned earlier, all important features of the video along with audio will be transcribed to preserve the closest connection between the two. Obviously we subsequently encode all transcribed data: meta-comments, comments or remarks by the experimenter; every piece of data is taken into consideration for the sake of an accurate theory. This leaves us with field notes and memos written during the experiments and analysis phase. Unfortunately, not much was written down while experimenting, but because both audio and video was captured, we claim that this data will suffice in writing post hoc memos. This process will be explained and developed in further detail in section 6.5.

Another data source – which is often overlooked – is the background of the experimenter. Because human research is not neutral, one must incorporate analyses of the effects of the background of the researcher [1: 261]. *Reflexivity* is the recognition that the product of

research inevitably reflects some aspects of the milieu, predilections, experience, education and many more properties of the researcher. Explicit formulation of these preconceptions and the underlying epistemology is crucial for objective results to emerge [6]: 91-93]. Nevertheless, we will not code this data but only include an account of the colouring aspects of the researcher involved. The outcomes of the research then can be interpreted further by the reader.
Selection of participants is discussed in section 4.1; selection within each of the chosen protocols is treated in section 4.2.

6.3 How to code

6.3.1 Focal points

While coding, there are several questions one could ask oneself about the data to improve the quality of the codes under development [6: 41, 1]:

- What is going on?
- What is the subject doing?
- What does the above imply / what inferences are used by the subject?
- When does this happen, and under what conditions?
- What are the consequences
- How do the structure and context serve to support these actions and statements?

Following these guidelines, the resulting encoding vocabulary has a broad coverage of phenomena and is at the same time on an analytic level.

6.3.2 Interpretation of segments

There are two difficulties on separate levels that arise while coding [4]:

- How to resolve ambiguities in interpretation?
- How much context to consider in the interpretation?

Both questions relate to the amount of context needed in order to resolve uncertainties and ambiguities in interpretation of a segment. Despite the fact that segmenting itself delivered fairly isolated units of information, they are only context independent to a certain level. The broadness of context allowed to use therefore needs to be determined.
The use of a broad context could improve reliability of encoding, but may bias the encoder towards expected operations (hence making inferences for the subject, which are not verbalised. Consequently, the distance between analysis and actual data increases) [5: 289]. The advisable

approach is to keep the interpretation at a fairly local level. However, this holds only when using multiple subjects, as the availability of more data then obviates the extra noise generated [4]. Either way one should always be consistent in terms of how broad a context to consider throughout a specific coding [4].
Because of the segment size applied, we do not expect interpretation to be a problem when investigating the segments to assign codes. Nonetheless, when there is no unequivocal way of interpretation, we allow ourselves to include one preceding and one following segment in the act of code construction and assignment as well.

6.4 Encoding vocabulary

After deciding what information is allowed to use in interpreting the segments, we have to think about *how* – by what means – we will interpret the segments and what codes to assign to capture its meaning best.

6.4.1 Data-driven or concept-driven?

First: should we use a coding scheme that is data-driven, or one that is concept-driven? According to Ericsson and Simon, encoding decisions should be based upon the task model, the theory under test and/or the problem space. They are clear that the vocabulary should be explicated prior to encoding to avoid data contamination by ad hoc theory [5: 264, 7]. On the other end of concept-driven encoding exists common, "predeveloped" coding schemes. Some attempts have been made to design a coding scheme that can be tailored to suit various research studies. Results indicate that such general-purpose encoding schemes could be helpful as template to determine the basic structure of a coding scheme [7].
On the other hand, supporters of the GT method strongly advocate data-driven coding. By letting the codes emerge from data, they say, the data will come to its rights best. Because no predefined codes are then to be forced upon the data, the researcher remains open and as unbiased as possible towards discoveries of new patterns or unexpected processes. In addition, several practices of research in the field of Software Engineering support the data-driven approach as well [7].
Nevertheless, it is impracticable to start the analysis with a complete *tabula rasa*: especially novices with GT need a perspective to be able to distinguish relevant data from the irrelevant and to abstract significant theories.

If desired, the codes and categories of pre-existent frameworks together with outcomes of a task analysis can be compared to the emerged codes. This could eventually lead to better, more analytic codes, implying enhanced inter protocol comparability [6]. However, due to time constraints, such a comparison will be skipped – something that clearly should not be left out in successive research.

6.4.2 Analytical or descriptive codes?

Next is the question what type of codes – what words or short phrases – we will use. It is acknowledged that there are several advantages to use a somewhat direct encoding scheme to capture the units and structure of verbalisation [5: 309]:

- It is easier and more reliable to encode;
- Automation of encoding is achieved easier;
- More information can be encoded.

Direct encoding vocabularies will consist of codes derived from the transcription in a most straightforward – hence reliable – manner. However, to get at a certain analytical level, one has to raise a level. An increase in analytical level results in an increase of abstractness. As a consequence, it becomes harder to reliably find codes that suit what the data tries to convey. Unfortunately, this is the price that has to be paid for improved inter protocol compatibility and a wider range of vocabulary applicability.

6.4.3 Size of vocabulary

Third, there is the size of the vocabulary under development. There exist some guidelines on the size of an encoding vocabulary in order to maintain its reliability and ease of use: when vocabularies grow too large for instance, it is likely to contain many similar codes. This increases the risk of applying different code to analogous events [5: 206]. The amount of codes has no magical limit but should be kept to a minimum to keep analysis coherent [11].

6.4.4 Codes per segment

Forth, in order to capture the behaviour to the fullest extend, multiple codes could be assigned to a single datum [9]. Since behaviour is composed of various components, each component should be covered with at least one code.

6.4.5 Reliability

Finally, reliability and consistency should be assessed. Typically this is done by the use of a second encoder [7], but in our case we have no access to such resources and neither have we the time to employ such practices. We assert that reliability can be assured by constantly comparing the already encoded transcripts with the transcript currently being encoded. A technique called flip-flop – i.e. moving back and forth between raw data and conceptualisation – aids with this. *Flip-flop* (or *constant comparison*) is the process of updating the codes emerged from previously encoded transcripts with new data [9, 6, 1: 193]. A decrease in correspondence between the data and its model then is duly noted and action could be taken to bring it back at an acceptable level.

However, mere encoding reliability[7] is not enough: multiple encoders could make the same faulty steps in encoding, leading to the same but incorrect result [5: 290]. Task analysis and/or predeveloped coding vocabularies could increase reliability, but only at cost of decreased creativity and open-mindedness.

6.4.6 Pitfalls

Designing a code vocabulary and relations between the codes has several benefits, including initial analysis of the data by providing insight into relations among the codes and a better overview.

However, care should be taken when interpreting data: interpretive steps are susceptible to misinterpretations induced by, for example, bias of a researcher towards one solution [6: 43]. The use of computer programs could aid during the analysis, but the actual work still needs to be done with human hands and heads [1: 233]. In order to minimise a possible bias, segments should be encoded in random order, which prevents the encoder from using previously encoded segments. Because professional encoders have experience in encoding work and probably prior knowledge of the experiment, they might be biased towards the hypothesis of the experiment and may expect the subjects to think same as themselves. More information about this reflexivity can be found in section 6.26.2.

[7] *Encoding reliability:* An encoding is reliable when multiple different transcripts are encoded in similar ways using the same labels for segments containing similar elements. Inter coder reliability and intra coder reliability refer to respectively reliability between coders and reliability within one coder.

6.4.7 Our implementation

As described, we stay close to the GT approach and apply data-driven coding: at the risk of decreased reliability, we remain open to every (unexpected) pattern in behaviour. Every change in the emerging encoding vocabulary will be allowed, and as we advance, the vocabulary grows and shrinks until all data is covered. Subsequently, the amount of codes will be reduced by removing duplicates.

Unfortunately, due to time constraints, triangulation of the emerged vocabulary with a task-analysis will be skipped. Also, segments will not be encoded in random order: the "segmenter" and encoder are both the same person and regardless of the encoding order, bias will occur. Nevertheless, an account of the researcher will be provided.

6.5 Memoing

During the process of coding, researchers look into the underlying meaning of each transcript. Coding compels them to think about the data, to make interpretations and look for connections. This provides insight into what is happening and at the same time concretises ideas about these processes. It is important to write those ideas down into memos, store them and be able to access this information in the upcoming stage of analysis. Writing memos is accounted to be *the* fundamental process in generating a grounded theory through which the researcher analytically interprets the data: segmenting and coding are mere tools to structure the data in order to ease the development of new ideas and interpretations. Memos are meant to conceptualise the data, to produce an abstract account of the phenomena, rather than to describe them [1: 245].

During the processes of transcription, segmentation and coding, we will write memos to structure our thinking, to record our ideas, hypotheses and remarkable phenomena: they are the account of ourselves talking to ourself [1: 119 & 249]. As literature on GT describes, they will often be messy, incomplete and consisting of only nascent ideas at first. Nevertheless, as the process of growing a grounded theory progresses, they will become increasingly clear and consistent [1: 249]. To be thorough, memos about memos will be written and treated like data as much as the encoded transcriptions [1: 258].

The main characteristic of memoing applied here relates to the emergence of codes and how they are saturated. In the later stages these memos will be used in writing more extended memos to correlate the codes to other codes and properties of the particular answer. These so-called *extended memos* will be regarded more as *drafts* of the first research report. It is them that will evolve into more concrete, better structured and more expounded accounts of the results of the research.

6.6 Summary

The aim of initial coding is to capture the detail, variation and complexity of the source data [9]. The initial descriptive codes provide indicators for the more abstract codes ultimately aimed at [1: 272]. Throughout this process the researcher gets acquainted with the data and establishes preliminary understanding of what is happening: patterns begin to emerge. Recognition of these patterns will give the researcher confidence in the coding process and their own creativity [1: 276]. Concurrently, he partakes in the act of memoing to aid saturation, and structuring of codes. The emerging codes become less descriptive and more analytic and, as a consequence, saturate [6: 46]. Complete saturation is reached when the vocabulary fully covers all variations in the data. When – after some time and effort is spent – the emerged codes cover the data to a satisfying extend, one is ready for the subsequent phase of categorising.

7. How to categorise

7.1 What is categorising?

Categorising is the analytic step in GT in which certain codes are selected that appear to have more importance than others or are more abstractly describing the common themes in data. During categorising, the researcher is obliged to look at relations between the codes with the aim to raise the level of concept reached with coding from descriptive to analytic. The combination of codes and relations amongst them results in categories that help the researcher in constructing a better theory or hypothesis [3: 186].

7.2 How is it applied

The codes that came up during the previous phase have to be organised into groups or subcategories and related to one another. In GT a hierarchically ordered structure of subcategories can develop from which a few core categories ultimately emerge. As with coding, categorising has its own process of saturation. Much analysis and reanalysis is required to verify categories – especially the core categories – through saturation, relevance, and workability [1: 280]. Those relations among categories and subcategories are regarded as theoretical properties of the raw data as opposed to the perceptible intrinsic properties [1: 196]. Researchers then are encouraged to find major themes by comparing the initially found categories and determining its relevance [1: 194]. Such a major theme is to be called a core category, which relates to as many other categories and their properties as possible, and it accounts for a large portion of the variation in a pattern of behaviour [1: 280].

7.3 Inherent difficulties

Similar to memoing, this process of categorising is more high-level than coding. The researcher's ingenuity is pushed to its limits in order to gain creative insights and come up with unapparent relations between different behaviours. It is because this resemblance that the memos of the previous phase are very helpful in devising the categories. Even more than with coding, developing categories is no algorithmic task: it relies purely on insight and creativity of the researchers.

One should be wary of overlap in categories, as that results in unnecessary intricacies and ambiguities, hence thwarting reliability [5: 208]. Also, an overlap increases the number of categories which has the same negative results as with coding (see section 6.4.3). Around five to ten categories seems to be a reasonable aim, depending on the sample size [5: 206].

8. Constructing grounded theory

The finishing step is of course the construction of the grounded theory itself. However, not much is left to be said because most of the work is already been done in the previous phases. However, there still is no theory. What is still in need of our exertion?
During coding, categorising and its concomitant memoing, structure in the raw data became apparent. One or more core categories have emerged and memos contain all information necessary to form an extensive theory.

8.1 What to do?

The next step is to write the research thesis, which is more than just mere reporting: writing and rewriting are crucial phases in the analytic process of writing down a theory [3: 154]. Compelling arguments for the theory at hand are to be manufactured. The researcher should ask himself: "So what? What is it about?". Sections containing strong arguments should answer this question without hesitance [3: 156]. Through sorting the memos following his own scheme, even more cross joints become visible. The emerging sorting then could serve as basis for the developing grounded theory [1: 120]. Again, memos serve as the fundamental link between data and the emergent theory [1: 249].
Besides the writing, literature review is an important process of this last phase as well: knowledge of a wide range of disciplines enhances a researcher's ability to see his emergent fit to a developing theory. Reading from both the research domain and other disciplines opens the researcher to serendipitous discovery of new theoretical codes, categories and relations from other disciplines. The more open one is to recognising the larger integrative patterns around us, the more one can exploit their imagery in proposing theories of behaviour [1: 283]. The extensive use of literature also enables a researcher to remark gaps in theorising and helps him see whether something discovered is interesting, useful or even new [1: 254].

8.2 Contributions and intended result

Of course, at first, the contributions of the theory under construction to the specialty field might not be as clear as one would like, but theories can prove their relevance because the method of inquiry is relatively new to the area under investigation [3: 153]. New methods of experiment and analysis pave the way for other researchers by opening their eyes to other previously unapparent aspects of their research domain: the theory should shed light upon a phenomenon from a different angle than the existing body of research already did. Furthermore, it should contain intrinsic value as well: theories should be falsifiable in order to have any scientific value [2, 10]. More falsifiable theories describe more aspects of the inquired in a broader sense and hence are better. Because of that, good theories should be plainly put and precise: the more intelligible a theory is, the easier and more accurate its effects are reproducible, and the better it could be tested [2].

The result of the whole process, from inquiry to writing, is a theory that theorises about the meaning of actions and relations between them [3: 151]. The emergent theory is vividly described and cogently underpinned with the use of the written memos accompanied by an introduction and conclusion. Besides that, extensive literature review from the field under investigation as well as other domains back up the constructed grounded theory. The whole is an argument fully based in empirical observations [9].

9. Threats to validity

Most threats to validity were discussed alongside the appropriate topics. However, few subjects are in need of some elaboration, more than previously accounted for.

9.1 Falsifiability

Because GT is data-driven, some adversaries of GT claim that theories devised in such manner do not contain any scientific value: they object that, when rejection should occur because of a fundamentally disparate occurrence, grounded theorists do not even renounce the theory; they simply modify it. Broad coverage then is guaranteed, but it then cannot distinguish the correct from the faulty incidents.
As with every inductive method, eventually high falsifiable hypotheses should be formed; in GT however, this should only happen when reached fairly advanced stages of the theory building process [1: 212].

9.2 Experience

GT is a difficult method of experiment and analysis: novice researchers often underestimate the dedication necessary to come to interesting results. The skill of a grounded theorist is to abstract concepts by leaving the detail of the data behind, lifting the concepts above the data and integrating them into a theory that must explain, not merely describe what is happening [1: 272-273]. Such skills develop while applying GT. Hence, inexperienced researchers often do not come up with as striking new theories as the seasoned researchers. Nevertheless, when applied correctly, at least no incorrect theories are constructed.

10. References

[1] BRYANT, A., AND CHARMAZ, K., Eds. *The SAGE Handbook of Grounded Theory*. Sage Publications, 2007.

[2] CHALMERS, A. F. *What is this thing called Science?*, 3rd ed. Open University Press, 2008.

[3] CHARMAZ, K. *Constructing Grounded Theory*. Sage Publications, 2006.

[4] CHI, M. T. H. Quantifying qualitative analyses of verbal data: A practical guide. *Journal of the Learning Sciences 6*, 3 (1997), 271–315.

[5] ERICSSON, K. A., AND SIMON, H. A. *Protocol Analysis: Verbal reports as data*, 2nd ed. MIT Press, 1993.

[6] GIBBS, G. R. *Analyzing Qualitative Data*. Sage Publications, 2007.

[7] HUGHES, J., AND PARKES, S. Trends in the use of verbal protocol analysis in software engineering research. *Behaviour & Information Technology 22*, 2 (2003), 127–140.

[8] KWINT, B. How do we look at uml? BSc. Thesis at Leiden Institute of Advanced Computer Science, August 2010.

[9] LYONS, E., AND COYLE, A., Eds. *Analysing Qualitative Data in Psychology*. Sage Publications, 2007.

[10] POPPER, K. R. *The Logic of Scientific Discovery*. Rootledge Classics, 2002.

[11] SALDANA, J. *The Coding Manual for Qualitative Researchers*. Sage Publications, 2009.

[12] YUSUF, S., KAGDI, H., AND MALETIC, J. I. Assessing the comprehension of uml class diagrams via eye tracking. *International Conference on Program Comprehension 15* (June 2007), 113–122.

Appendix C

The Active Processing Assumption

Transcript of participant 21, coded according to the colour scheme introduced in section 6.1 of the accompanying thesis.

Example
EQ
P
[T] [D] [p1] Ok [p1] So I'm looking for a task execution, so I'm looking on the right side through the [p1] graphics. I see [p1] task [p1] schedule, log, so I prefer it's a history call so I'm going for a log. [T] On the left paper I'm just scanning.. [p1] Is there a time limitation or something?
E
There's no time limitation.
P
Ok. Then I'm just reading it once, [p1] slowly. {mumbles while reading} [p3]. I found something again: [p1] logging? [p2] Ok.. [p3] I'm looking for the history right?
EQ
P
Ok. [p3] [D] [p1] [T] The problem is in the text it tells me it's batch result table [p3] or the batch [D] log [T] table. So [p1] I guess [D] it's the [T] log table [p1] in the database [p2] the history.
E
The log table?
P
It's not quite [p1] just guessing.
E
The log table.
P
[p2] Yes. [p2] Oh, no, sorry [p1] "The additional logs can be found [p1] The batch result table is the basic [D] answer
E
The batch result table?
P
Yes.

Architecture 1: Alpha
EQ11
P
[D] Personal details, source [p1] [T] external [D] [p1] Ok. [p1] [T] [p2] [D] [p1] [T] [p3] [D] [p1] [T] I think it's the [p1] Paraplu-system.
E
Paraplu system?
P
Yes.

EQ12
P
[D] [p2]
E
Try to think out loud if you can.
P
[p1] I see at the graphic [p1] on the left it's a search [p1] has to be a structure it works [p1] with the integration service. [T] I'm back in the [p1] document on the right [D] [T] There was something [p1]

before the preparation interface service [D] is the [p1] [T] headline of the footchapter. It's a proxy service [p2]
E
Proxy service?
P
Yes.
E
Ok. Third question ..
P
[D] {Says something}
E
Sorry
P
No..

EQ13
P
[D] [T] [D] [p1] Ok, [p1] I'm looking again at the [p1] graphics if there's something like searching. [p2] Webservice [p1] There is something called search repository hidden down on the left. [p1] Person data, Paraplu [p1]
[T] I'm looking for [D] words in the text like search repository [T] or repository. [p2] [D] [p1] [T] {Mubles text} [p2] Last sentence: "In this way Paraplu will be search based on the application profile." [p1] [D] [p3] [T] [p3] The question was how does Paraplu [D] search? Or..
EQ13
P
[p1] [T] [p3] [D] [p2] It works all with the preparation interface system. [p1] It gets, [p1] I have to look at the graphics for this one, it goes out and up [p1] to the left preparation interface system which gets down [p1] the search repository [p1] of the Paraplu data webservice personal data.
E
Ok.

Architecture 2: Beta
EQ21
P
[T] [p1] [D] Ok. [p1] I look at the [p1] graphics first. [p1] It's a bit [p1] unstructured, so I'm starting somewhere in the middle [p1] bus messages, no [p1] it's not related [p1] manage use database, user database, [p1] private [p1] webinterface sounds a bit like [p1] security [p1] webfrontend, [p1] here's authentification, there's another one which could sound like.
[T] I'm going back to the text [D] because the [p1] paper is not good [T] [p1] So, I'm looking for [p1] eight [p1] numbers [p1] something written out, but [p1] It's not clearly, so I'm starting again [p1] Enterprise, intercommunication provides service, [p2] straight connections are not possible [p1] only via webfrontend.. [p2] Login names must be unique and passwords must be between 6 and 9 characters, so eight characters would be good.
E
Ok.

EQ22
P
[D][T][p1][D] The what users?
EQ22
P
Authenticate, [T] ok .. [p1] [D] [p1] Looking at the graphic webfrontend webinterface. [p2] It says [T] it generates the bus message, so it's unclear.. [p3] So in the text it states that in the middle [p1] that its [p1] enterprise message bus takes care of it. [p3] It's working on the authentication system which is connect to user database. [p1] Ok. [p1] So, [p1] message [D] I think [T] would no [p2]
E
No?
P
No.

EQ23
P
[T] Ok [p1] [D] So again, [p1] external system is on the graphic [p1] request availability [p1] is on top left [p1] that's generates bus message: it's all [p1] connect somehow to the [p1] enterprise bus. [p1] So, [p1] external [T] request system is the thing [p1] "All components can be collect [p1] that [p1] accept something that .. {mubles text}" [p2] To verify, can you [p1] repeat the question again?
EQ23
P
Availability. [p3] Right now I'm just scanning the document for availability or anything which could give me clues.. [p3] There's no [p1] special thing about this one. [p1] Messages can be ignored is a general.. [p2] It's unclear about this one so I guess yes, [p1] they can be ignored.

Architecture 3: Gamma
EQ31
P
[D] [p2] Ok [p1] The graphic looks quite good for it, [p1] it's nicely strucured [p1] I think.. [p1] Backoffice system is easy to find, so
[T] I will look in the text.. [p1] [D] One of the [p1] [T] services. [p3] Ok, so [p1] there's nothing about, the text about big about it's. I have to [p1] [D] look at the [p1] graphics only.
So, [p2] client data administration let me put it everything in the graphic basically. [p2] Is this as an answer to you..?..
E
Yes? Ok.

EQ32
P
[T] [p1] [D] [p3] No.
E
No?
P
No.

EQ33
P
[T] [p1]
E
Try to think out loud if you can.
P
Yes, I'm [p1] [D] I read this [p1] before [T], so I'm just looking. I think it was 300k. [p1] I'll have to check. [p1] "Decreased backoffice, mortage actions,.. requests may have a maximum size of 300kB" [p1] So, [p1] yes, this could [D] be a limitation. [p1] I just read before, just remembered somehow.. [p2] It's not really in the graphics so [p1] [T] in the text, 300kB [D] [p1] [T]
E
So your answer is yes?
P
Yes.

Architecture 4: Delta
EQ41
P
[D] [p2] So, in the graphic, [p1] it's [p1] very nice [p1] nicely structured, [p1] on the left is [p1] management server, it's using SOAP.. [p2] So, [p1] billing system, {mumbles} [T] ok, [p1] back to the text [D] [p1] [T] booking for management server [p1]. There's no management server in the headline [p1] so I have to search for it. [p3] So, [p1] there's something .. [p1] management [p1] no. [p1] Normal read. [p3] Question again please?
EQ41
P
[D] [p1] [T] [p2] Billing information [D]
E
Billing information.

P
Yes.

EQ42
P
[D] [p1] Ok, I'm back in the right page. I'm looking for the [p1] schedule creator [p1] thing. [p2] {mumbles text} .. "get their credits .. [p1] using SOAP to access the webserver." [p3] [T] Question again [D] please?
EQ42
P
[p1] [T] The problem is here [p1] the definition 'node'. I can [p2] .. {mumbles} .. before [D] I guess it's the [T] webserver [p1] but I'm unsure.. [p2] {mumbles 'node', as if he's looking for that word in the text} [D] [T] [p2] [D] It says [p1] it communicates [T] through the webserver [D] by the means of SQL. [p1] So I guess it's webserver.
E
Webserver?
P
Yes.

EQ43
P
[D] [p2] [T] [p3] [D] [T] [p1] I think no.
E
No?
P
No.
E
Ok.