UNIVERSITY OF OSLO
Department of informatics

Personality and Pair Programming: How do Pair Programmers Collaborate?

Master thesis
60 credits

Thorbjørn Walle

7. May 2009

Table of Contents

Preface

Activities similar to pair programming are involved in many of my earliest memories of computer use. When playing the computer games of the early nineties (which came in large boxes, filled only with maybe a thin manual and several diskettes), my friends and I often used a procedure where one person was controlling the keyboard and mouse, and the other person suggested future moves and alerted the first of incoming enemies and such. Periodically, the roles were switched. This technique was especially useful and interesting for both players when we were playing adventure games (point-and-click non-text-based adventures was the hot new thing). In these games, reasoning was more important than reflexes and the actual controlling of the character, and there were rarely situations where quickness mattered. Often, the person who owned the computer would yell at the other person because the latter, the "navigator", would point at the screen, suggesting some place to go, or item to use. And in the days of the 15", 30lbs CRT monitor, this was especially annoying due to the large and visible finger smudges which would appear on the screen. "Hey, don't touch the screen! I thought I already told you" was a phrase often said and heard in those days.

Although possible, LAN and internet gaming was still totally unknown to most people, including myself, so the above procedure was as collaborative as gaming could get. At least it was a lot more collaborative than taking turns playing, as the standard was in many early "multiplayer" games. In the case of the classic Super Mario Bros, player one would be Mario and play first. When he "died", player two, controlling Luigi would play until the same happened to him. Then it was Mario's turn again, and so on.

Another situation where computer use was often quite collaborative during my early computer years was the making of poorly animated movies. These were made by using very basic animation, and later; simple 3D software, such as the classic Microsoft 3D Movie Maker. Mostly, these movies were little more than five minute remakes of movies and series famous at the time, such as Jurassic Park, Godzilla and MacGyver. For this animation work, pair programming-like techniques were especially suitable. One person was usually more used to the controls of the animation software, and the actual work of doing the animation was not much fun either. However, when deciding the plot (which we, of course did along the way, with no script or plans ahead, since we were young and bold), both of us were equally active. This was also the case in the voice-over work and when taking minor decisions regarding the visual aspects of the scenes and their flow.

Later, the project assignments at grade school where we used a computer turned out similarly. Certain people were the fastest keyboard typists or the most skilled word processors. These were naturally often placed at the keyboard. The others looked through encyclopedias, which we would then make poorly described references to. The non-typists also checked continuously for typos and other problems and suggested wordings of the sentences, but the one at the keyboard was most often the one in *de facto* command.

Not until I enrolled at the university did I try any actual programming, though I had tried different types of scripting for games and such, as well as html and some custom queries in the database platform MS Access. Remarkably, despite my good experiences with pair programming-like work patterns from an early age, I never thought of trying pair programming until working with a really large development project in my third year. In this, we were subtly encouraged to pair program, and both XP and pair programming were topics for the lectures in the course. In this project we were supposed to make a system based on java code generated by high-level UML and GUI editors. None of my project members considered themselves above average java programmers, and the task was, compared to other assignments in University courses, very complex. Even so, we did good work on it, and were commended for having produced one of the best systems that semester. As we learned later in the course, the situation of being inexperienced/junior programmers combined with working on a complex problem was actually the very same situations as was shown in a large and recent pair programming experiment to be the situation most suitable, and beneficial, for pair programming.

In the evaluation of factors of success and failure, we recognized pair programming as a major success factor. From that course on, I was a pro-pair programming person, even though I did realize the increased effort involved. I suggested using pair programming in several projects later, and we often did.

Because of my good experiences with pair programming, and my interest for basic psychology, I was eager to sign up for this masters assignment about pair programming, collaboration and personality traits as soon as I heard about it. I have never had any regrets about that decision. Pair programming can be very useful in certain situations, but it is still a somewhat disliked practice among developers. A better understanding of in which situations to use it, and when it is not an apt technique would not only prevent it from being used wastefully and in vain or wrongfully discarded, but could also make people stop disliking it as a matter of principle. Pair programming is not for everyone, and it is certainly not for all situations. However, for some situations and some combinations of people, it is great.

Introduction

For investigating the collaboration involved in pair programming, we coded 44 audio recordings of pair programming sessions. We divided the audio recordings into codeable segments, and assigned each segment to categories that were selected from a collection of categories we had developed specifically for use on these recordings.

After we had coded the audio files, I performed a literature review. I then formulated several propositions for possible relationships between the personality of the pair programmers and the collaboration during the pair programming sessions. I then did a statistical analysis in order to investigate these propositions.

Although most of my propositions turned out to be unsupported by the analysis, my major propositions were proven to be valid. My most important proposition investigated whether or not personality had any impact on pair programming collaboration at all. The analysis for the proposition showed that nearly all personality factors affect the occurrence of at least one collaboration category.

Another important proposition, the one which was most closely related to findings from the literature I reviewed, was also shown to be apt. Pairs consisting of two people with different personalities will communicate more than the pairs where the two programmers were more similar to each other. However, we saw that not all personality trait differences affected this relationship in the same way, so a high difference in every personality factor will not necessarily be beneficial for the collaboration.

All in all, it is clear that the personality of the two programmers will significantly affect the collaboration during pair programming; but for now the impact that the different kinds of collaboration has on the performance of the pair programming pairs is not as closely investigated, and it will be subject of future research. Several of the propositions analyzed yielded insignificant but promising results, and could be conclusively proven, or perhaps disproved, if the analyses on them were done with a larger sample size.

1. Problem Statement / Research Question

Does personality affect the collaboration of pair programmers? is the overall research question for this thesis is. More specific research questions can be found in Chapter 12.

Below is a brief description of the master thesis as it was proposed by the supervisor, followed by a quick look at the experiment on which the thesis is based, and the article in which the first results from that experiment was presented.

1.1 The Masters Thesis Problem

The problem was proposed as a long masters thesis, entitled “How do Pair Programmers Collaborate?” It was proposed by Jo Hannay at the Department of Software Engineering at Simula Research Laboratory, and was formally administered through IFI (Department of informatics) at the University of Oslo. The thesis description, as stated on the IFI website, was as follows:

> “Pair programming is a way of programming within the paradigm of eXtreme Programming. Pair programming involves two programmers collaborating over one keyboard on the same programming task; a situation that inspires a particularly close form of collaboration. The thesis/theses will investigate how pair programmers collaborate, and will further see if there are any links between form of collaboration and the personality types of the two members in a pair. For example, whether extroverts talk more, whether conscientious people have more task-focused conversation, and whether people with low emotional stability have more conflicts in collaboration.“
>
> ”In the first step, you will analyze audio recordings of actual pair programming collaboration using structured analysis techniques such as conversation analysis and negotiation analysis. This first step will serve to categorize the ways the pairs collaborate into predefined categories of collaboration types. In the second step, you will assist in investigating links to the personality types of the members of the pairs.” [IFI 08]

1.2. Collaboration Involved in this Thesis

As well as continuous contributions from my supervisor and contributions from other people associated with Simula, the collaboration coding schema was developed in collaboration with one other student. This person works on a similar thesis, but the focus of her thesis is the programmers’ expertise levels; contrasted to my focus: personality.

The work with coding of audio files, as described in Chapter 9, was also done in cooperation with this same master student, plus additional people that were hired by Simula Research Laboratory.

Everything else was done individually. Collaborative work directly inserted into the thesis is Appendices X1-X4. All other parts of this written thesis were not a collaborative effort.

1.3. The Pair Programming Experiment

1.3.1. Description

The audio recordings to be analyzed in this thesis were sampled during a previously conducted large pair programming experiment, whose main focus was to compare the performance of pair programming to that of solo programming [Arisholm 07]. The experiment was conducted in 2001, 2004 and 2005 and involved a total of 295 junior, intermediate, and senior professional Java consultants (99 individuals and 98 pairs) from 29 international consultancy companies in Norway, Sweden, and the UK. The Java consultants were hired at market price for one full work day and were selected to give three levels of expertise: junior, intermediate and senior. The pair programming material, which is relevant to this thesis, was collected in 2004 and 2005.

Statistics about the subjects' expertise level (based on the consultancy companies rating and their performance in a pre-test) were gathered, along with information about each person's experience and attitude towards pair programming. Personality information about all subjects was also collected; each participant in the experiment completed the "big five" personality test. The two members of each pair had a level of programmer expertise (junior, intermediate or expert) similar to each other. Most pairs had no experience with pair programming.

During the day of the subjects' participation in the experiment, the subjects were to perform six tasks, four of which were meant to be analyzed by the experimenters. The two first tasks were performed individually by all subjects. The first task was a training task to familiarize the subjects with the experimental environment, the second task was a pre-test task used to measure the skill level of the subjects when working alone (regardless of whether they would pair program or not in the later tasks). The four last tasks were performed individually (solo programmers) or in pairs (pair programmers). These four tasks were all connected, since they all were change tasks on the same system, and the solution of one of these tasks was the basis of the next task. Thus, it was important for the programmers to solve a task correctly, since it could be harder later on if they did not. The very last task was a time sink task (meant to make sure that no subjects rushed a solution in order to leave early), and the results from this task were not used in the experimenters' analysis.

The subjects were about equally distributed to two slightly different versions of the experiment. One was a system with a centralized control style with a powerful central class who did most of the system's functionality. The other was a system with a delegated control style where most classes had their own responsibilities [Arisholm 07]. The latter was more similar to the object-oriented ideal that Java encourages. Otherwise, the two systems were equal, and for users, which would only use the functionality, and not read the source code, they would appear like the same system. This difference in control style was important for the primary use of the experiment, but not directly relevant in the collaboration analysis for this thesis. However, knowledge about which control style the pairs were working on was essential for the analysis of collaboration, since the pairs working on the centralized version talked about different things than did the pairs working on the delegated version. These differences were especially noticeable when the pairs discussed system flow and mentioned names of components of the system, which varied quite heavily between the two versions of systems. It was important to understand to what they were referring from the very beginning of the task in order to analyze the dialog correctly.

The conceptual model and hypotheses behind the experiment are described in great detail in the article, and unusual terms used in the article are explained. The design and execution of the experiment is also thoroughly documented. Large sections of the article are technical and contain statistical calculations and results, and the statistical methods and packages used are also referenced and described sufficiently. The article describes a number of threats to validity, both internal and external threats, and how they handled them. The article is published in the renowned "IEEE Transactions on Software Engineering", indicating that the experiment, and the description of it, is recognized as significant and valid in the scientific community.

2. Descriptions of the Chapters

2.1 Introductory Chapters

Chapter 1 is a description of the problem statement for the thesis and the experiment on which the thesis work is based. Following this is a review of previous related and semi-related studies and articles.

Chapter 3 is a definitions chapter. First, we give some in-depth descriptions of major concepts that recur in the text and that are central to the investigations. Directly after this are short and concise definitions of other concepts and how they are to be understood when used in this text. The available literature uses many different words for some of the same concepts. Therefore, alternative variants of the concepts are mentioned, in order to make it easier to relate concepts from various sources of literature to each other and to this thesis.

Chapter 4 describes the main objectives for this thesis. It also contains a presentation of related literature.

2.2 Coding Chapters

The chapters are about the coding of the audio files. First, in Chapter 5, ideas and considerations when making a coding schema are mentioned, as well as an introduction to protocol analysis.

Chapter 6 is a review of some articles containing coding schemas that are of special interest. Most of them are either about pair programming, or personality and collaboration.

The development history of our coding schema is given in Chapter 7. It is described chronologically, so the newest version, which was the one that was actually applied to the audio files, is the last one described, and is described in Section 7.7. The different versions of the coding schema are described in detail, with special attention to describing differences from one version to the next, and how well the schemas performed during their testing. Inspirations/references and reasons for the changes are also described. The complete final coding schema and definitions for its categories are shown in Chapter 8.

Chapter 9 describes the process of applying the codes to the audio files, and how we extracted analyzable data from this process. Although this step was the part of the investigation that produced most of the results, the work here was mainly repetitive and quite straightforward once the coding schema and coding decisions were final. For this reason, the chapter is short compared to the time spent on this step.

Even though the coding schemas were tested continuously during their development, there was a need for a final and more thorough reliability control at the end of the coding in order to make sure that the results was scientifically significant and not overly subjectively based. The process of this check is described in Chapter 10.

2.3 Research and Results Chapters

Chapter 11 describes overall matters regarding the analysis.

Chapter 12 contains research propositions for possible relationships between personality traits and collaboration. The first research proposition is based on findings in the reviewed literature. The last propositions are based on the definitions of the personality traits and the collaboration categories.

Chapter 13 describes the considerations and procedure of the statistical analysis based on the data that we extracted during the coding of the audio files.

In the ensuing chapter, Chapter 14, the detailed results of these analyses are listed.

2.4 Ending Chapters

In Chapter 15 some possible threats to the validity are listed. Chapter 16 lists possible future directions for continuation of this investigation and possible hints and considerations for closely related new studies.

At the very end, a selection of appendices can be found. Most of the appendices are earlier versions of the coding schema and descriptions of them.

3. Definitions

3.1. Pair Programming

> Pair programming is a style of programming in which *two* programmers work side by side at one computer, continually collaborating on the same design, algorithm, code or test. One of the pair, called the *driver*, is typing at the computer or writing down a design. The other partner, called the *navigator*, has many jobs, one of which is to observe the work of the river, looking for tactical and strategic defects. Tactical defects are syntax errors, typos, calling the wrong method, and so on. Strategic defects occur when the driver is headed down the wrong path–what is implemented just won't accomplish what needs to be accomplished. [Williams 06]

The two roles are often illustrated by using the allegory of rally racing. The driver controls the car, while the navigator reads the map and tells the driver about upcoming turns and obstacles and the situation regarding the opponents ahead of and behind them.

It should be noted that pair programming is a term with a slightly misleading wording:

> We use the word *programming* to include all phases of the development process (design, debugging, testing and so on), not just coding. So, pair programming would include pair design, pair debugging, pair testing and so on. In fact, two studies has indicated that pairing is most important for analysis and design […]. We believe people should pair at any time during development, in particular hen they are working on something that is complex. The more complex the task, the greater the need for two brains. [Williams 06]

This last statement, that complex tasks are especially suited for pair programming, is elaborated on in Section 3.1.1 below.

Pair programming is one of the twelve practices of the agile software engineering methodology eXtreme Programming (XP) of the late nineties and 2000s. However, pair programming can just as well be used when using a more rigidly structured or sequentially based development process such as the widely used waterfall model. Also, pair programming is not a new technique invented and introduced from scratch by the agile community. Williams [Williams 06] describes different people's experiences with pair programming in the seventies through nineties, and even mentions an author claiming to have used the technique as early as in the mid-fifties.

3.1.1. Claimed Advantages

The well-known and thorough, although noticeably pro-pair programming, book by Williams and Kessler [Williams 03] claims that pair programming leads to several benefits: The code quality will be better, i.e. the code will have fewer defects. The time it takes to solve tasks will be about half compared to individual programming, thus nullifying the double development cost that will occur because of the two programmers compared to the usual one. Pair programming is also claimed to have other advantages, such as in the areas of morale: Pair programmers are stated to be happier programmers. Further, the authors say that pair programming increases trust and teamwork between employees, since they get to know each other through the activities of pair programming. Knowledge transfer will increase, especially if people occasionally or often pair program with different people, i.e., so the pairs are not the same all the time. Finally, pair programming increases learning since the two programmers learn from each other about development and cooperation. [Williams 03]

Many of these benefits, however, are contested in articles and experiments. As both the experiment in [Arisholm 07] and a large systematic review of pair programming effectiveness [Dybå 07] and [Hannay 09b] reports, the benefits are more obvious in certain situations and less so in other situations. The effectiveness is shown here to depend on factors such as the expertise of the programmers and the system complexity. Both articles state that though the quality/correctness is noticeably higher for pair programmers, the cost/effort is also much higher.

3.1.2. Effects

In the large pair programming experiment [Arisholm 07], junior programmers benefited the most from pair programming in terms of quality (+73%), but did not solve the tasks any faster than they would have if they had worked individually. Duration is indicated as +5%, which indicates that pairs actually used *more* time than individuals. Intermediates and seniors benefited little from pair programming in terms of correctness: Intermediates scored 4% better and seniors actually scored lower in correctness when pair programming: -8%. However, their task duration was somewhat reduced. For intermediates, the number was -28%, for seniors: -9%.

The authors also compared results between solving of easy and complex problems. For easy problems the duration was reduced when using pair programming (-20%), but so was the correctness (-16%)! For complex problems, the duration was slightly increased (+6%) for pair programmers and the quality was moderately increased (+48). Juniors solving the complex tasks benefited greatly in terms of correctness (+149%) when pair programming, while the seniors benefited little from it (they actually performed worse) in terms of quality no matter how complex the system was [Arisholm 07].

3.1.3. Critiques

Critiques of pair programming are often directed towards the effort/duration involved. Many claim that pair programming is too expensive; since two people are doing a task that one person could do alone just as fast and as well. They believe that pair programming is a waste of time for all situations other than training [Williams 03].

Many have more personal arguments against pair programming, and dislike it because they prefer to work alone. They might dislike the fact that credit (and blame) is divided, and they think it will be hard to be maximally efficient when working with others [Williams 03].

3.2. Big Five Personality Traits

For a long time, there has been an interest among psychologists and others to define specific personality traits and divide people into groups based on these. Not all of these trait systems, however, have been scientifically based. Much of their contents remain un-validated and not recognized as apt by others than the inventor of the particular classifications.

The “Big Five”-system was meant to be a contrast to those, and was an attempt to make a personality classification system that would be scientifically sound and based on empirical findings.

> In 1981, Goldberg reviewed the work of others, as well as the results of his own research. Impressed with the consistency of the results, he suggested that “it should be possible to argue the case that any model for structuring individual differences will have to encompass –at some level–something like these ‘Big Five’ dimensions” [Pervin 01].

The *big* in Big Five "was meant to refer to the finding that each factor subsumes a large number of more specific traits" [Pervin 01]. The *five* indicates that there are five of these factors: Extraversion, agreeableness, conscientiousness, neuroticism and openness (to experience). Each trait is bipolar, for example, if you score low on extraversion, you automatically score high on introversion, since that trait is on the other end of the same scale. Each trait is described further in the subchapters below.

As stated in [Hannay 09a], "Big Five" and "Five Factor Model" (developed by Costa and McCrae) are commonly used as synonyms. The factors themselves are the same five in both, and they are similarly defined, but the two systems differ in that they have slightly different foci. Goldberg's "Big Five" is what was measured and used during the experiment, and what we will be talking about in this text.

3.2.1. Extraversion

Sometimes called *surgency* [Raad 00] or *dominance-submissiveness* [Cloninger 04], extraversion "assesses quantity and intensity of interpersonal interaction; activity level; need for stimulation; and capacity for joy" [Pervin 01]. As stated above, the bipolar trait of extraversion is introversion.

De Raad [Raad 00] claims, in a reference to Guilford & Braly, that "No single pair of traits of personality has been quite so widely discussed and studied as that of extroversion and introversion". In De Raad's book, extraversion is defined as

> […] the outward turning of psychic energy toward the external world, while Introversion refers to the inward flow of psychic energy towards the depths of the psyche. Extraversion is denoted by habitual outgoingness, venturing forth with careless confidence into the unknown, and being particularly interested in people and events in the external world. Introversion is reflected by a keen interest in one's own psyche, and often by a preference to be alone.

For extraversion, a low scorer will typically lack interest and maybe ability to socialize a lot or make large networks of acquaintances. This can lead to barriers in getting ahead in the social and professional life, where recommendations and getting introduced to the right people are important factors in many situations. However, the low scorers will have a lot of time for themselves that can be used for intellectual pursuits, hobbies or work.

Both the comical "geek" stereotype and results from the experiment suggest that computer programmers are generally low scorers in extroversion. It is not unthinkable that this is related to their interest and ability in the technical pursuit of making or manipulating computers and their software. Either their interest for computers is so big that they prefer this to socializing, or they dislike socializing so much that they avoid it, and have chosen programming or other hobbies as alternatives.

Keirsey [Keirsey 08], who operates with an extroversion scale similar to that of the big five, mentions that several historic people have been introverts, including top scientists like Niels Bohr, Stephen Hawking and Isaac Newton and political figures like the 19th century British Queen Victoria and American presidents Truman and Eisenhower.

3.2.2. Agreeableness

Agreeableness, sometimes referred to as *social adaptability* or *likeability* [Cloninger 04] "assesses the quality of one's interpersonal orientation along a continuum from compassion to antagonism in thoughts, feelings, and actions" [Pervin 01].

The bipolar trait of agreeableness is mentioned as being antagonism [Pervin 01]. People of low agreeableness are probably those who will be most unfavorably depicted in today's society. However, one should not assume that people of low agreeableness are necessarily "bad people". Since agreeableness, as the other five factors, is a composite of several sub-factors, low scorers might have some of these factors high, but receive a generally low score because of a radical low score in others. The free big five test at personalitytest.net, divides agreeableness into the following: Trust, morality, altruism, cooperation, modesty and sympathy. If a person is very moral and modest, he will rarely do anything to harm or diminish others, and will not act in unethical ways to increase his own status, fortune or other factors of life quality. However, he will still score low on agreeableness in total, if he is not a "people person", and consequently receives low scores on trust, altruism and sympathy.

3.2.3. Conscientiousness

Sometimes referred to as *dependability*, *impulse control* and *will to achieve* [Cloninger 04], Conscientiousness "assesses the individual's degree of organization, persistence, and motivation in goal-directed behavior. It contrasts dependable, fastidious people with those who are lackadaisical and sloppy [Pervin 01].

The bipolar trait of conscientiousness is "lack of direction", according to [Pervin 01]. Conscientiousness is a trait for which a low score has quite a significant impact on work performance on nearly all kinds of work [Barrick 01].

3.2.4. Neuroticism / Emotional stability

With its bipolar opposite often referred to as *emotional stability* and sometimes *emotional control and ego strength*, [Cloninger 04], neuroticism "assesses adjustment vs. emotional instability. Identifies individuals prone to psychological distress, unrealistic ideas, excessive cravings or urges, and maladaptive coping responses" [Pervin 01]. Cloninger mentions that "people who score low on neuroticism are happier and more satisfied with life than those who score high."

One must keep in mind that a low score of emotional stability in relation to the Big Five is not an indication of mental disorders or insanity. Emotional stability is simply a trait of the personality which indicates that the person has more expressed emotions and/or urges. It does not describe how "mentally stable" a person is.

3.2.5. Openness to experience

Called *culture*, *intellect* [Raad 00], *intellectual interests*, *intelligence* and *imagination* [Cloninger 04] in alternative naming of the category, Openness (to experience) "assesses proactive seeking and appreciation of experience for its own sake; toleration for and exploration of the unfamiliar" [Pervin 01].

Though alternative names for the trait and certain descriptions of it might indicate that it is related to the intelligence or mental capacity of people, it is not. Like the other traits of the Big Five, it is merely a measure of behavior preference.

4. Pair Programming and Personality

Pair programming is a highly collaborative technique involving a lot of discussions, questions and answers, advices and other types of teamwork-related conversation. The form and quality of this collaboration will depend on many factors: How difficult the problem is and how skilled the programmers are is important. Whether or not the two programmers have worked together or know each other from previous tasks or personal relationships can affect it. How the circumstances around the work are (i.e. how much time they have, how comfortable the chairs are etc.) is also a possible influencing factor. Lastly, but certainly not least, the personality types of the two people in the programming pair is likely a very important factor in this.

As seen in Section 3.1.3, a recurring critique of pair programming as a development technique is that it is a waste of time, since the navigator finds only syntax mistakes, which could be found by compilers just as well as, or better than humans any way [Williams 06]. While this may be true in some situations, the role of the navigator includes much more than just checking the spelling and syntax and working on the level of the written code. He is supposed to also think ahead and envision the system as a whole [Williams 06]. To investigate whether or not people of a certain personality profile will focus more or less on syntactic issues, will make us able to know for which pairs the problem of syntax-only focus of the navigator might most often occur.

4.1. Previous Related Research

In the past, there has been some research on personality in connection with pair programming, among others in [Hannay 09a] and [Sfetsos 06]. However, the focus of this research has primarily been to investigate a possible direct correlation between personality and the performance of the pair. Sometimes other factors such as nationality, gender, expertise and task complexity has been included in an attempt to grasp the bigger picture of pair programming and what affects it. Most of this has, however, concluded that there is just a minor correlation between the personalities of the people in a pair and that pair’s performance.

One of the largest and most recent studies on this [Hannay 09a] states in the section for possible future research that to include collaboration as part of the relationship between personality and performance could possibly lead to new and more significant findings.

Figure 1 illustrates the difference in conceptual models between the formerly conducted studies and the indirect approach for correlation between personality and pair programming performance. As one can see, the latter used with collaboration used as mediating factor, and is the one which will be investigated in this thesis. It is possible that the personality of the programmers of the pairs will influence how that pair collaborates and that collaboration in turn will have an influence on how well the pair performs, even though personality had been shown not to heavily influence performance directly.

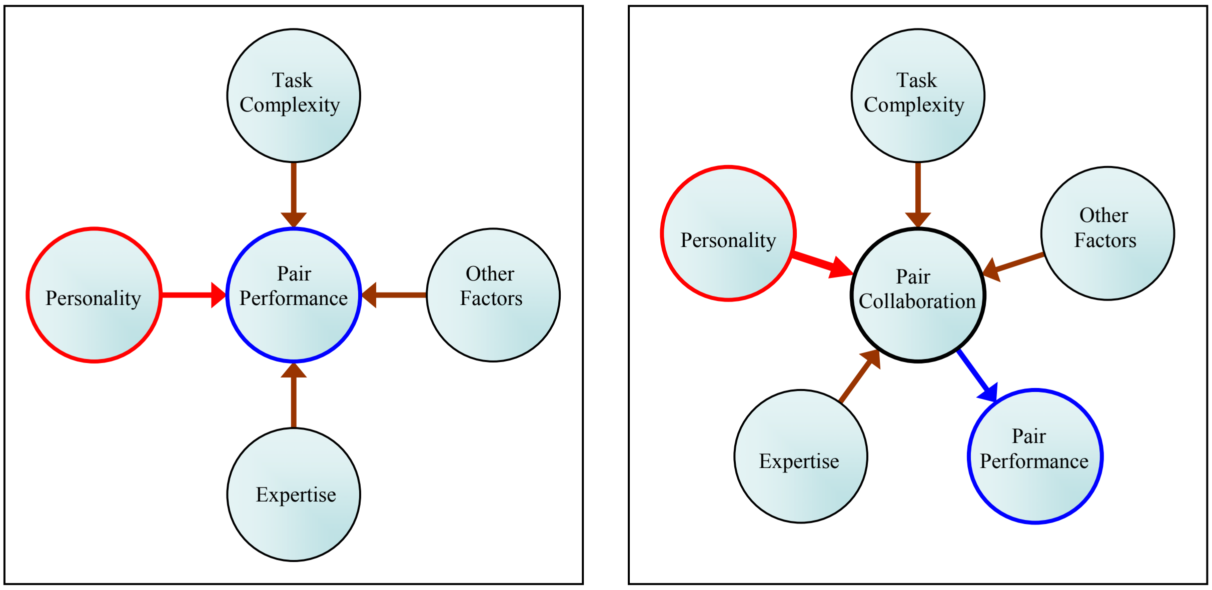

Figure 1: The two conceptual models for investigating different factors' influence on pair performance

Most of the previous research on pair programming and personality differ from this new one also in the personality classification chosen. The studies that are mentioned in the review in [Hannay 09a] used either the quite well-known Myers-Briggs Type Indicator (MBTI), a custom generator made by the researchers themselves (sometimes based on others, that they modified), or other lesser known classifications.

"Big Five", (which is described in Section 3.2) is based on empirical research [Goldberg 93] contrary to the MBTI, which is not. Because of this possible strength of "Big Five", it is possible that the personalities will be better described by using this, and that the correlation between the programmers measured personality and the factor it is measured against will be greater.

The following sections contain summaries of the related literature that was reviewed for this thesis.

4.1.1. Hannay, Arisholm, Engvik and Sjøberg

During the big pair programming experiment described in Section 1.3 a lot of information about the programmers, the tasks they did, and the programming results were gathered. For this reason, the experiment could be the basis for a quite high number of different analyses with a variety of foci.

Soon after the completion of the experiment, researchers at Simula started investigating relationships between the different factors in order to investigate how and when pair programming could be used for ideal results. System complexity and programmer expertise were the first two factors to be investigated [Arisholm 07], but personality was soon to follow [Hannay 09a].

In the latter of these two, the authors conclude that "personality traits have a modest predictive value on pair programming performance compared to expertise and task complexity" and that "Rather than focusing on direct effects of personality issues on performance, [...] more effort should be spent on elaborating the nature of collaboration and its effects on performance". Larger differences in extraversion and agreeableness between the two members of pairs were found to lead to worse performance scores, but in a lesser degree than all the non-personality based variables: task complexity, programmer expertise and programmer nationality.

The article also contains a summary of related work. Some of the articles from it, in addition to some others, are discussed below.

4.1.2. Dick and Zarnett

To find out what personality traits an employer should look for when hiring people for pair programming, Dick and Zarnett [Dick 02] interviewed eight developers, including themselves, after having pair programmed with each other for a period of around three weeks. They asked each developer to state traits they thought to be important for successful pair programming. The traits the authors present in the article as the results of these interviews are "Communication", "Comfortable", "Confidence" and "Compromise".

One can wonder if it was a coincidence that all traits started with a "C", or whether some of the traits they actually found are more commonly known under another word, but changed here to make it catchy.

The article explains what is done and concluded in an understandable way. The results are based on discussions during interviews rather than quantitative statistical calculations. The article and its results are not based on Big Five, MBTI or any other personality trait system one could more or less scientifically test people on, and is dependent on a job interviewer that manages to investigate these four areas during the discussion.

While the article cannot be said to answer exactly what we like to know, since it does not focus on the personality traits' influence on pair programming collaboration, the article presents some interesting findings that will be of use for our hypothesis development later.

4.1.3. Karn and Crowling

Karn and Crowling perform [Karn 05] an ethnographic study in which they sought to find correlations between the MBTI personality values of teams consisting of 4-6 software engineers and how these teams collaborate and perform. The authors do a similar investigation one year later [Karn 06]. The articles follow three groups of students working on three different projects.

They claim in their results sections that homogeneous teams (referring to homogeneous in personality) are not ideal and that they "run into a real danger of falling into the no debate trap" [Karn 05]. Furthermore, they state that, contrary to their assumptions, disruptions are not always bad for the collaboration. Disruptions are rather stated as being beneficial for collaboration, since disruptions lead to more discussion and dialog. In their follow up of the study [Karn 06], however they say that the revisit made it clear that disruptions were, in the long term, damaging to the development teams.

The article's findings on the advantages of having non-homogeneous personalities in groups, and the effects of disruptions, will be of good use in our hypothesis construction phase. What limits this article's usefulness to us is that the collaboration the authors refer to in the article is a collaboration total for all tasks. Of these tasks, management, especially group meetings, constitute the largest part. Also, in this article, the quality of collaboration is measured by counts and types of disruptions only.

4.1.4. Williams, Layman, Osborne and Katira

In an article [Williams 06], these authors present an investigation in which they try to investigate which factors will make a pair programming pair *compatible*. Their definition of a compatible pair is one where both programmers perceive the pair to be compatible. The compatibility score is thus not measured by performance or anything else, but only a subjective statement by the subjects.

Regarding personality, the authors test the hypothesis that “Pairs are more compatible if students with different personality types are grouped together”. Another of the article’s hypotheses is similar, but concerns learning styles, referring to a scale for this that contains the sensing-intuitive-scale of MBTI as one of its factors. For one phase of their three phased study, both these two hypotheses receive what the authors refer to as “partial support”. That is, personality is proven to be a predictor for when pairs evaluate themselves as very compatible.

The article is detailed and describes research with a large sample size, and in which many interesting findings and results are presented. Although it does covers personality and performance, it does not state any specific personality traits that are desirable for efficient pair programming or how the collaboration will be affected. The article does, however mention what kind of personality traits one should look for if the goal is to make the two people in the pair pleased with the other pair member.

As stated in Hannay et al. [Hannay 09a] “[…] 93% of pairs (randomly allocated and irrespective of personality) report compatibility”. This large number might suggest that the measure of compatibility was imperfectly defined or measured.

A related article, written by some of the same authors was published two years before this one [Katira 04]. Its focus is slightly different, but the results are similar (that mixed personality types are best for compatibility). This older article will not be discussed in detail due to its similarities with the newer one.

4.1.5. Barrick, Mount and Judge

Unlike the other articles that were reviewed for this thesis, the article of Barrick et al. [Barrick 01] actually discusses the Big Five personality classification. On the other hand, the focus is not on pair programming. Indeed, it is not even on software development.

The article is a review and summary of 15 articles regarding personality traits and job performance. Their findings suggest that conscientiousness and emotional stability (the inverse of neuroticism, see Section 3.2.4) is correlated with good work performance. Although not really that close, the work category most similar to pair programming of those they present is “teamwork”. For this category also, a correlation with extraversion and agreeableness is shown.

The article and the review process therein are based on statistics and quantitative facts. Regrettably, it is of little use regarding our research questions, and our specific focus on pair programming. The idea of pairs or teams of any kind is not discussed in the article, and the focus is on the personality of individuals and their performance in a job only. This is not a weakness of the article, since combinations of people and personalities are not among their focus areas. However, it does make the article less relevant for us. The results and discussions from the article will be used during our hypothesis construction, but it will hardly be the lone source of any hypothesis we will investigate.

4.1.6. Sfetsos, Stamelos, Angelis and Deligiannis

These authors write in an article [Sfetsos 06] a description of a set of experiments where the objective is to “compare pairs of mixed/heterogeneous developer personalities and temperaments with pairs of the same personalities and temperament, in terms of pair effectiveness.”

The subjects, eighty four undergraduate students, were students who were measured on their performance on the “Cockburn’s Responsibility Drive coffee machine code”, which is not unlike the coffee machine used in the experiment in [Arisholm 07]. They measure the programmers’ personality by using MBTI and the “Keirsey Temperament Sorter” (KTS).

Their results indicate that pairs with mixed personalities performed better at the tasks. Relevantly to this thesis, the authors also state that the mixed personality pairs also “communicated better” [Sfetsos 06]. Communication is measured with a very high-level focus, and based on the navigator keeping a log of conversation, rather than third parties analyzing the actual conversation and collaboration.

This study is very interesting for us, since it its experiment has many similarities to the one this thesis is based on. The article also focuses on communication and collaboration. Its methodology, however, is not that similar to our. The findings in the article are highly interesting, so it will be very useful to us for hypothesis construction and for getting ideas and inspiration.

4.1.7. Choi, Deek and Im

This article [Choi 08] discusses an experiment with students as subjects where the authors investigate whether pairs of alike, diverse or opposite personalities perform better. They focus on the two middle factors of the MBTI system, which indicate how a person interacts with and perceives the world. They present definitions for personality similarity and lack thereof. An *alike* personality would mean that a person has those two middle letters similar to the partner in the pair. The two could for example be INTJ and ENTP and be considered alike. *Diverse* means that one of the two middle factors is dissimilar and *opposite* means that both are different.

They conclude that the diverse pairs are the best performers, the opposite pairs perform moderately well, and that the *alike* pairs are suboptimal. The article describes the experiment and the statistical measures thoroughly. However, like many of the other articles; it is concerned only with performance, and not collaboration. The findings are a little too specific in a field not similar to ours to help us in the hypothesis making phase, but the article will provide inspiration and ideas.

4.1.8. Chao and Atli

Their article [Chao 06] describes a survey in which the authors tried to identify the five personality traits most commonly perceived to be beneficial for pair programming. The result is that pair programmers should be open-minded, creative, logical, responsible and attentive. Open minded sounds both in name and definition similar to the Big Five trait openness. Responsible seems to be quite similar to the trait of conscientiousness. (See Section 3.2 for definitions of the Big Five traits.)

In the second phase, they run an experiment in which they test how programmers of different combinations of these five traits perform (that is; the traits they found in phase one: not the "Big Five" personality traits). Their results are stated as being not statistically significant, but pairs with two low scorers in either open-minded or responsible scored the lowest.

The article says little about collaboration and focuses mainly on programming performance results, which makes it not that relevant for us. Its use of a custom classification of personality also makes it quite incompatible to our study. However, its findings make some suggestions about which personality traits that lead to good performance, and could be of use in other situations.

4.1.9. Others

Lucas Layman's article "Changing Students' Perceptions: An Analysis of the Supplementary Benefits of Collaborative Software Development" [Layman 06] was also reviewed, but deemed irrelevant for our study, because of its focus only on attitudes and the change of it. Therefore, it will not be described in detail here. Its results include that

> "Personality type (MBTI) and learning style (FSLS) had little effect on attitude change. Students who disliked collaborative experiences were predominantly reflective learners, introverts, and strong coders." [Hannay 09a]

Lynda Thomas et al. wrote a similar article [Thomas 03] to Layman's, but without focus on personality types. It concludes that confidence (i.e. self-confidence) is important both for enjoyment and results of pair programming. Because of its lack of focus on collaboration and other elements of personality, it will not be discussed further. An article by Brian Hanks [Hanks 06] is similar, and focuses also on confidence. Its results are non-significant and not really relevant to us, so it will not be described further here.

An article by Karim Visram [Visram 04] slightly touches the subject. It is a strictly theoretical paper, which claims that "successful pair programming […] relates to Goleman's traits of Emotional Intelligence" [Hannay 09a]. Since this is not a focus of the research in this thesis, however, Visram's article will not be discussed in detail here.

Not only in recent articles is personality mentioned as an interesting focus area for research for programming. Shneiderman mentioned in his 1980 book that "Personality variables play a critical role in determining interaction among programmers and in the work style of individual programmers. Unfortunately too little is known about the impact of personality factors." He continues by mentioning several personality factors that are considered as relevant for programming: Assertiveness, extraversion, motivation, tolerance for ambiguity, compulsive precision (which he defines similarly to the big five trait conscientiousness), humility, tolerance of stress, and locus of control. The latter is a psychological term which he defines with the following example:

> Individuals with strong internal locus of control feel able to and seek to dominate situations. They feel they have the capacity to influence their world and control events. Individuals with external locus of control feel that they are victims of events beyond their control and are perfectly content to allow others to dominate them. [Shneiderman 80]

However, pair programming, or even team based programming, is not the focus of his attention, so even though these and other ideas he presents is a good inspiration for what to look for, little of it will be of any direct value for the analysis of collaboration in pair programming.

Audio File Analysis

Hughes and Parkes [Hughes 03] mention that an obvious way for collecting information about what people are thinking during tasks is to perform interviews on the subject when they have completed the task. However, they then quote Durkin who states that “At worst, interviews and questionnaires can produce inaccurate reports, due to erroneous or distorted reporting: a person's recollection of his or her thinking may be significantly different from the actual thoughts made at the time”. It is feasible that a fair amount of people will, deliberately or not, describe their thoughts and performance more favorably than in reality, since this distortion of the truth would be nearly impossible to detect. If detected, it could easily be excused by saying “I forgot” or something similar. If people did this, the material based on their accounts would obviously not be good for analyses, since this material would not represent the truth. Note that this is a positivist opinion. People with epistemologies more commonly associated with the social studies might disagree.

5. Protocol Analysis

A protocol, as the concept is used in this thesis and the related literature, is a recording of speech and/or actions done by people, where someone wants to keep this kind of record. The recoding can be in the form of a transcription text, audio recordings or something else, such as video recordings. An example of a protocol, based on this definition, is the permanent record that is sometimes transcribed during proceedings and criminal cases in courts of law. In our case, the protocol is the verbal contents of the audio recordings we did, i.e. what the programmers said during the experiment.

Ericsson and Simon [Ericsson 93] are two of the most prominent people in protocol analysis research, and have literally "written the book" (actually several books) about the subject. They further emphasize the view described above, stating that:

> A critical problem with […] verbal descriptions of […] cognitive processes […] and experiences is that such reports do not relate clearly to any specific observable behaviour. Even where subjects are asked to report in their cognitive processes used during many trials of an experiment, we cannot rule out the possibility that the information they retrieve at the time of the verbal report is different from the information they retrieve while actually performing the experimental task. To avoid this problem of accessing information at two different times – first during the actual cognitive processing and then at the time of report – we proposed that whenever possible, concurrent verbal report should be collected, so that processing and verbal report would coincide in time.[Ericsson 93]

To think aloud during the tasks is their suggested solution to the problem.

Verbal protocol analysis has, according to Hughes and Parkes [Hughes 03] "been prominent in cognitive research psychology for over 25 years". Usually, this was done by analyzing what people said to themselves when requested to, or voluntarily, "thinking aloud" and explaining to themselves or others what they were doing when they did it. Hughes and Parkes claim, based on a reference to Ericsson and Simon [Ericsson 93], that "thinking aloud will directly represent the contents of the subject's working memory". It is also stated that "the technique provides a bias-free method of revealing what a person is thinking when performing a task."

Protocol analysis is evidently useful for getting insight into thought processes and how a programmer/developer does things, for example if the goal is to examine how they use a new tool or a new development process. A downside, however, is that for most people, it is unnatural to speak to themselves. If forced to, they might not say everything they are thinking, and in the worst case scenario they could omit the very information that is most important to the researchers. This might especially be the case if the deepest thought processes are the ones that are investigated. It is likely that many people, if forced or asked to "think aloud", would say only what they were certain of, and just the concrete things they do ("Now I'm writing the next function call"). The questions they subconsciously ask themselves, and their decision making ("It will be easiest to use an iterative function here, since I know that I don't understand recursion so well") could be what the investigators were most interested in, but a fair share of people would probably keep this to themselves.

In pair programming, however, discussing and elaborating on nearly all programming-related issues when they appear on the fly is the natural way to act. It is often necessary to explain your thoughts and courses of action for others to understand them even if the task at hand is well known for both parties. Protocol analysis is thus a very useful and applicable method for analyzing any aspect of pair programming. It is perhaps especially good for analyzing one of the most important differences between pair programming and individual programming; the collaboration involved.

For this thesis, the medium of analysis was already defined: audio recordings from the large pair programming experiment in [Arisholm 07], each containing one pair programming pair and their discussions when solving the tasks. Therefore, since we had the available "think aloud"-like verbal protocols available, protocol analysis was the way to go. But there were still quite a few decisions to make, including what part of the dialog that would be of most interest to focus our attention upon, and how, and what, statistical data could be gathered from our analysis of audio files.

5.1. How to Do It

To analyze audio recordings, several methods can be used. Myers, in his overview of qualitative research [Myers 97], mentions two modes of analysis. The first, hermeneutics is described as being concerned with the meaning of what is said. The other, semiotics "is primarily concerned with the meaning of signs and symbols in language" [Myers 97]. He divides the latter further into three sub forms. Two of these are *content analysis* ("looking for structures and patterned regularities in the text") and *discourse analysis*.

> Discourse analysis […] focuses on "language games." A language game refers to a well-defined unit of interaction consisting of a sequence of verbal moves in which turns of phrases, the use of metaphor and allegory all play an important part. [Myers 97]

The latter, discourse analysis, seemed like the most suitable method when evaluating collaboration. How the pattern of the conversation is going (Who is talking when? Do they get an answer? Do they interrupt each other?) is an interesting area of attention when measuring collaboration. However, elements of content analysis will also be helpful to investigate what the programmers are discussing and when, and if pairs with certain characteristics tend to talk more about certain things than others.

5.1.1. Making the Protocol Analyzable

The verbal protocols were, in our case, lengthy and numerous. In order to be able to produce statistics, and more importantly, sense, of the protocols, some sort of classification is needed. We considered that what was said and/or what was done was the obvious focus of attention. We would divide the verbal protocol into sections, and give each section a code based on what was happening or what was said during the clip. These codes would be one or several categories chosen from a coding schema we developed. The goal was to select the categories that described the activities and the type of collaboration that was present during this section of the file. We referred to these sections to be coded *clips*. How long the clips should be was an important matter. In examples from the literature, the clips they coded could range from as little as single words (in the extreme cases even parts of words if the tonality change during long words) to as large sections as several minutes' discussion about a topic.

Hughes and Parks [Hughes 03] summarize the process as follows: to perform verbal protocol analysis, one must first collect raw data (verbalizations, like audio recordings), then segment and encode it, before finally analyzing it. They also state that for the segmentation and encoding, one should use the same categorization and rules for all subjects in the same context. This is obviously an important rule, and one we will follow.

Regarding coding schema development, Hughes and Parks quote Chi [Chi 97]:

> However, the process of devising a coding scheme can be quite challenging for a researcher. Although an initial coding scheme may be constructed from the literature, published research is unlikely to provide a perfect model which can be re-applied: the choice of codes will depend upon the subject domain, the hypotheses being tested, the research questions being asked, and the 'theoretical orientation' of the researcher (Chi 1997).

Bakeman and Gottman's book Observing Interaction [Bakeman 97] can be considered a manual of coding schema making. They stress that a good coding schema which is fully compatible with the material of research is of utmost importance, and creation of it should not be done in a rush.

> The first step in observational research is developing a coding scheme. It is a step that requires a good deal of time an attention. Put simply, the success of observational studies depends on those distinctions that early on become enshrined in the coding scheme. [...] Yet sometimes the development of coding schemes is approached almost casually, and so we sometimes hear people ask: Do you have a coding scheme I can borrow? This seems to us a little like wearing someone else's underwear. [Bakeman 97]

5.2. Considerations for Making the Coding Schema

5.2.1. Physical or Social?

In their book, Bakeman and Gottman [Bakeman 97] mention that one should be aware that categories can be physically based or socially based. In our context, physically based categories would be regarding what is done, i.e. programming and reading, and socially based categories would be questions and answers and other dialog-based categories.

Bakeman discusses this further, and explains about facial expressions, body language, etc. Since we are to analyze just audio, rather than also video, facial expressions and other body language will not be possible for us to know anything about. Moods will for the same reason also be omitted from our focus. While it would be possible in many situations to identify what kind of mood people are in from their audio expressions, it is more often impossible to determine this, and often people can show their mood visually without speaking. This will, as stated before, be unavailable for us with our audio files only. Also, analysis of facial expressions, body language and moods would be a very time consuming process, and it would most likely require some psychology background or at least a lot of specific training for the coders to be able to do correctly.

Our categories turned out to be a mix of both physical and social categories, but with most focus on the social part.

5.2.2. Objectivity

When developing a coding schema, one must consider how objective in use one wants to make it. A coding schema as simple as counting each time a person says yes and each time a person says no, would be possible to use with great objectivity. On the other hand, a coding schema where it was required of the researcher to distinguish between whether or not a given disagreement was "solved" or not, would require more insight, and in some situations subjective thought.

The obvious advantage of having the schema maximally objective is repeatability. Two different coders would with a 100% objective coding schema, if assumed that they do no errors, get the exact same result. For a heavily subjective schema, the result would depend on the coders' considerations, and would most likely not be completely identical.

Regrettably, many aspects of collaboration are hard or maybe impossible to measure objectively without breaking collaboration down to its most atomic and crude terms and code these. While it is possible to do it that way, for example by coding each and every word (like it is done in the article discussed in Section 6.3) it will be extremely time-consuming, and might in many cases lead to "robotic" results that is of little use in showing things about human action. Bakeman mentions that physical categories are not necessarily objective, and mentions as an example that it requires cultural knowledge in order to be able to code for example body language.

In the end, our categories turned out acceptably objective, while also being suitable for describing collaboration and it was not too "robotic" when applied to the protocol. A reliability check of the categories was done in order to ensure, among other things, that the objectivity was satisfactory. This is described in detail in Chapter 10.

5.2.3. Exclusivity

Bakeman [Bakeman 97] states that it is important for coding schemas to be mutually exclusive and exhaustive. "This means that only one code can be associated with a particular event (mutually exclusive) but that there is some code for every event (exhaustive)."

This was regarded as important in our category development, and new versions were often mostly based on better category definitions (exclusiveness) and extension of the list of categories (exhaustiveness).

5.2.4. A Priori or Not?

Hughes and Parkes [Hughes 03] review a number of research papers on the subject, and claim that "fewer than half of the research papers reviewed here used a coding schema which evidently had been prepared a priori".

Bakeman [Bakeman 97] quote Rosenblum and argues that the evolution of a code schema should be gradual and that the initial stages are best kept very simple.

> It is best to begin in the most unstructured fashion possible. There is great advantage to beginning such observations with only a pencil and blank pad for recording. [...] At the beginning, the observer must simply watch, allowing [...] behavior patterns to emerge as figures against the background of initially amorphous activity [Bakeman 97].

This procedure was followed in our research as well. In the beginning, the schemas were short, with categories which were not very specific. Coding would be accompanied with a log of events and description of why the categories were used.

5.3. Software

To help us during the coding of the file, we chose the software *Transana* as our tool. Transana is a "qualitative analysis software for video and audio data" developed at the University of Wisconsin-Madison Center for Education Research [Transana 08]. It is primarily meant for projects where real transcription is used, rather than just direct category coding of the audio, but it turned out to be highly was suitable for us also. Transana offers an easy way to make time codes and manipulate the flow of time in the files, since hotkeys makes it easy to rewind and stop when needed. One can easily export statistical data from Transana.

6. Relevant Existing Coding Schemas

Many schemas have been used or suggested in articles describing previously conducted studies. However, none of the many schemas which were available through literature searches completely fulfilled what we wanted in a schema. Therefore, we were required to develop and test our own custom-made schema.

Although we did not include any existing schema in its entirety into our schema, inspirations were gathered from several articles and schemas. Substantial parts of our final schema are directly based on categories from other schemas. In the next chapters, a selection of interesting and/or important articles is presented. After that, the chronological report of the development of our coding schema is described.

The articles are presented with special focus on how useful the classification were for us, when we developed our own schema, and how suitable it would be for measuring the collaboration involved in pair programming. Thus, even though some of the schemas are displayed in a negative light below, they are probably good when used in the situations described in the articles they are in.

The schemas below range from the very systematical and almost machine-like syntax in von Mayrhauser's coding schema [Mayrhauser 99] to the very qualitative and high-level classification used by Cao and Xu [Cao 05]. The different classification schemas all have their advantages and their disadvantages, and there is no universally appropriate schema that one can use for all situations.

The first two articles, discussed in sections 6.1 and 6.2 below, are about collaboration in pair programming specifically. The articles described in 6.6 and 6.7 are about pair programming, but they have no specific focus on collaboration. The article in section 6.3 is not about pair programming in particular. Instead, it focuses on the general area of software development. The "dialog" that is analyzed in the article described in Section 6.3 is meant to be "think aloud" protocol (See Chapter 5), but their schema is just as applicable for use on dialog. The article described in section 6.4 focuses on learning and mental models and is somewhat similar to that in 6.3.

The articles in sections 6.5 and 6.6, as well as those in 6.9 and 6.10 are not about software development or computer use at all. However, they do provide interesting findings regarding collaborative work and learning and what kinds of pattern that could appear in dialogues. How problems, arguments and problem solving are handled in groups is probably pretty much the same regardless of the type of task that is to be solved. The dialog would likely be quite similar in general structure for two people trying to solve a programming task and two people trying to direct a movie, given that the relative knowledge between the two when it comes to programming was the same as their relative knowledge of directing.

6.1. The Schema of Lan Cao and Peng Xu

This schema was presented in the article "Activity Patterns of Pair Programming" [Cao 05]. The article focuses on how collaboration will be different depending on the programming experience of people in the pairs. They mention that the pairs consisting of two people of similar experience levels are more successful than the pairs in which one member is a novice and one an expert. The authors also investigate the differences in the *activity patterns* for the different combinations of pairs.

The protocol coding schema in the article is stated to be based on two previously developed schemas. One of these two is the schema in Okada, et al. (described in section 6.10 below), which is very similar to Cao and Xu's schema in the detailed level of focus and the concepts used. The other one is Lim et al., which is described in more detail below, in Section 6.4.

Cao and Xu's schema focuses on quite abstract, high-level activities and dialogue patterns. Their main coding schema is shown in Table 1. In other parts of the text, they mention a few additional *sub-activities*, like "correcting misunderstandings" in the "Explanatory Activities" main category. As usual for this kind of schema, the categories are meant to be applied to sections of dialogue, which we refer to as *clips*. Frequently the clips used in the examples in Cao and Xu's article were quite large. Long dialogues consisting of several statements and responses are shown as examples of single codeable sections, as long as the subject discussed or the goal of what is said is the same throughout the section.

The remainder of their article contains explanations of the different categories and sub-activities, and examples of how to use them on specific sections of dialogue. Table 2 shows the closer descriptions of the first group of categories (as seen in Table 1), the "Leader's activities".

Categories	**Sub-activities**
Leader's activities	Set up goal/tasks Summarize current status Formulating Strategy/Action Reconcile differences
Ask for opinions	Ask for opinion
Explanatory activities	Request for explanation Provide answer with explanation or justification
Critiques	Modify solutions Raise new problems Explanations
Summary of results	Goal /task refinement Mental model extension

Table 1: Cao and Xu's coding schema

Leader's Activity	**Example**
Set up goals for the session Set up tasks for next step	I think we need to do is, the way we present: register, add, drop…right now we got a single class and based on that, we can send to student file.
Summarize current status	So once you add…you are going to add them in the database, and view and drop them, and add others.
Formulating strategy/Action	Now we're going to figure out how to pass these parameters.
Reconcile differences	So actually we can combine these two… so I log in to the faculty menu and student menu, and student menu will have these and you add these.

Table 2: Leader's activities

6.1.1 Advantages

The most apparent advantage of this quite simplistic high-level taxonomy is that it is fairly easy to apply to the segments/clips of the verbalization data. An experienced coder will be able to code the segments in near "real time" using this schema, since the categories are broad enough to be used on dialogue sections of several seconds' duration. If the coder manages to write and listen carefully enough at the same time, he will be able to write down categories for dialogue sections while keeping up with the pace of the dialogue. This implies that the schema could be used for coding pair programming as it happens. An audio recording of the pair's discussions would in that case, not be strictly necessary.

The categories are, though wide and abstract, concrete and precise enough to be applied to the dialog without the need for extensive amounts of subjective evaluation from the coder. The categories are mostly well-defined and understandable, so two coders trying to code the same section of dialogue, would most likely end up with a very similar result, given that they share a general understanding on certain coding issues mentioned in the disadvantages section below.

When the categorization is done, and the analysis is about to begin, this schema would provide the analyzers with quantitative data on how often each category was used. It would also be fairly easy to find out if certain transitions between different categories were common, (for example whether or not a request for opinions often lead to an explanation).

If one wanted, one could use this schema for focusing on finding results that are qualitative rather than quantitative. If the researchers did not focus only on the statistics of the category use, they could attempt to grasp the "big picture" of the results, which could lead to findings that one might not have found when analyzing just the category usage numbers. This would, however, require them to have extensive knowledge of the complete coding results, and this kind of qualitative analysis can rarely be successfully done automatically by a computer program or by people unfamiliar with the specific dialogues.

All in all, the schema is ambitious in its simplistic nature, and it might be excellent for use in certain situations. The authors, who possess complete understanding of the schema, were probably able to use it very successfully.

6.1.2. Disadvantages

One downside with the article is that it is never stated how long each coded section of the dialogue should be. Also, some of categories have slightly hazy boundaries, and might be indistinguishable from each other in some situations. For example, it might not be clear when the conversation should be coded as "formulating strategy/action" instead of "set up goal/tasks". Is it a "formulating strategy/action" at the first mentioning of the Java code itself? Or should this category be used as soon as the specification document is no longer discussed, or perhaps not until the actual coding has started and the first method call is done? Even the example given in the article (Table 2 above) to point out where the borders between these two categories go is a little unclear. A categorizer with an especially abstract mindset might code very long segments of speech as "set up goal/task", while a categorizer with a more micromanaging disposition might divide the section into many parts, and code these as "ask for opinion", "request for explanation" and others where appropriate. None of these two coders can be said to have done it wrong.

Certain dialogue patterns that are likely to appear in pair programming are not included in this schema. Categories describing disagreements and off-task discussions are lacking, and where these would be suitable categories, a category most similar to them must be used instead. This is not a big problem for us, however, since it is easy to extend the categories here without interfering with the others or destroying the "essence" of the schema.

Also, the categories say little about the collaboration, but rather focus on the activities. This is understandable, since it is the activity patterns which are the focus in the article, but it means that it is not all that relevant to us. It is limited how valuable the data we could find by using this schema would be, since we did not plan to perform a deep and probably time-consuming qualitative analysis.

6.1.3. Our Use of the Schema

The initial versions of our study's coding schemas were heavily inspired by this schema, but our final versions did not contain much of Cao and Xu's schema.

6.2. The Schema of Sallyann Bryant, Pablo Romero and Benedict du Boulay

This coding schema was presented in the article "The Collaborative Nature of Pair Programming" [Bryant 06]. The authors' goal in this article is to find out whether or not one of the pair programming roles (driver/navigator), or one of the people in a pair contributes more than the other to the dialogue.

Even though the authors do not focus specifically on the collaboration involved, they make a categorization that can be used for studying collaboration. The list of so-called "derived generic sub tasks" presented in the article is shown below in Table 3.

A	Agree strategy/conventions	Including approach to take, coding standards and naming conventions
B	Configure environment	Setting up paths, directories, loading software etc.
C	Test	Writing, running and assessing the success of tests
D	Comment code	Writing or modifying comments in the code
E	Correspond with 3rd party	Extra-pair communication: person to person, telephone or email
F	Build, compile, check in/out	Compiling and building on own or integration machine
G	Comprehend	Understanding the problem or existing code
H	Refactor	Re-organising the code
I	Write new code	Creating completely new code to complete the assigned task
J	Debug	Diagnosing, hypothesizing and fixing bugs
K	Find/check example	Looking at examples in books, existing code or on-line
L	Discuss the IDE	Talking about the development environment

Table 3: "Derived generic sub-tasks" from Bryant et al.'s coding schema.

6.2.1. Advantages

Like Cao and Xu's schema above, Bryant et al.'s categories focus on the high levels of programming, and it would be quite easy to use. Bryant et al.'s schema might be even easier to apply to the protocol than that of Cao and Xu, since the Bryant schema focuses on the activities being done, rather than the contents of the discussion. It is feasible that a coder will be able to code in real time when using this schema as well (as described in Section 6.1.1). The other two major advantages with Cao and Xu's schema also apply here; the schema is wide and abstract, but tangible enough to be applied to the dialog by most people. Also, most categories are well-defined

6.2.2. Disadvantages

As well as for advantages, this schema shares its *dis*advantages with the schema presented by Cao and Xu. Bryant's categories are even less suitable for describing collaboration. Also, the categories are very few in number, and they are all quite similar to each other. By itself, it would lack much if the goal was to describe the rich and varied area of pair programming collaboration.

6.2.3. Our Use of the Schema

This schema was used as a basis for the initial versions of our schemas, since it has a good mix of categories for different situations that will often happen during a pair programming session, such as testing. This line of categories "survived", and was included throughout each and every coding schema. In the final version of our coding schema, the inspirations from this article appear as many of the "task focus" categories.

6.3. The Schema of Anneliese von Mayrhauser and Stephen Lang

This coding schema was published in the article “A Coding Scheme to Support Systematic Analysis of Software Comprehension” [Mayrhauser 99]. The authors take a very technical position, and they begin the article with describing and explaining different mental models and actions involved in software development. This leads to a very systematic and detailed category hierarchy, where the mental model and other more abstract topics are at the top of the hierarchy, and specific actions are at the bottom. For example, the action of reading a list of variables is coded as SYS.act.inf.cod.rea.var, since it is program based (SYS), an action (act), “manipulation” of information (inf), based on the code (cod), done simply by reading (rea) and that what being read is a variable (var).

The article describes all the elements of the hierarchy in detail. Then the article goes on to explain what types of analyses which will be suitable to perform when using this categorization.

6.3.1. Advantages

This very technical, systematic and granular coding schema can, as mentioned in the article, be used for many analytic purposes. Analysis of the coded dialogue could even be done automatically by software. People who are not directly involved in the analysis and experiment could perform these analyses just as well as people who have a full knowledge of the experiment and the coding schema. The results of such an analysis would be a very quantitative and statistical report.

6.3.2. Disadvantages

The most important reason why we did not use this schema is the incredible amount of work one is required to do in order to use the schema. The amount of time required to code would most certainly be many times the actual length of the verbalizations. Also, for a fair share of people the coding process using this schema would probably be viewed as an exceptionally tedious and repetitive job.
It is also a concern that even though the categories are well-explained and very precise, it would be hard to use them in real life when coding dialogue. The example coding in the article shows that in many occasions, the coder (which is presumably one of the authors), has stopped progress downwards in the “code hierarchy” and just put up *:MIS*. This category is not explicitly explained in the article, but is used in a matter that suggests that it is short for miscellaneous. Also, certain sections of protocol are not given any category at all. It is reasonable to think that two people trying to code the same recording using this schema might end up with a noticeable difference in their coding results.

The categories focus very much on what is done, and not so much why and how it is done. Thus, it might be less suitable to use these categories to describe dialogue from pair programming sessions, especially when focusing on collaboration. In pair programming, discussions are often about long term plans and the understanding of the code, and not only discussions about code manipulation. Also, the action focus this schema has would mean that much of the important factors in collaboration, including the flow of the dialog, would not be covered by the categories.

6.3.3. Our Use of the Schema

The schema is very complex, and would require a lot of effort to use. For these reasons, the schema in its unaltered form was not seriously considered as usable for our investigation, other than during the very initial discussions when nothing was yet decided. However, some of the ideas presented in the article are useful. The system of using many subcategories in order to richly describe a segment was used in the later versions of our category development.

The *mental model* concept from this article was important to us, and became the basis for the mental model categories we included in middle versions of our coding schema. These were later renamed and altered to form the category class of "Cognitive Level" which was introduced during the later versions of our coding schema. The "Cognitive Level" category group remained a part of our coding schema all the way to the final version of the schema.

6.4. The Schema of Kai H. Lim, Lawrence M. Ward and Izak Benbassat

In the article "An Empirical Study of Computer System Learning: Comparison of Co-Discovery and Self-Discovery Methods", the authors seek to examine "two types of exploratory computer learning methods: self-discovery vs. co-discovery, the latter of which involves two users working together to learn a system" [Lim 97]. Accordingly, the focus of the article is learning. The article frequently discusses the concept of "mental models" which is described as the understanding a person has of the complete system and its details. The authors mention that a way to test a person's mental model is to ask them to imagine how a system would behave, given a specific input.

The categories in the coding schema try to grasp the "mental" dimension of teamwork. They distinguish between the two main categories: *deeper level reasoning* and *surface level reasoning*. Subcategories for deeper level reasoning are *Seeking Understanding/Clarifying*, *Reasoning/Inferring*, *Formulating Strategy* and *Evaluating/Interpreting*. For surface level reasoning, the subcategories are *Describing Action* and *Reading Screen Output*. Also included in the schema is a category for *Other Mental Processes*. The results presented in the article shows that co-discovery leads to a higher inference potential (the potential of generating proper predictions of a system's behavior), which in turn leads to better task performance.

6.4.1. Advantages

The categories in this schema are general and are focused on a quite high-level. The high-level focus might lead to the schema being easy to apply to the verbal protocol. The schema focuses on an interesting aspect that is mentioned by few of the other articles reviewed in this thesis; namely, the aspect of the understanding of the system and the development of the so-called mental models of the persons participating in a dialogue.

6.4.2. Disadvantages

The article focuses heavily on learning, which is a weakness when considering its usefulness for our study. While learning is the inevitable first step in order to improve a system, learning itself was not the focus of the pair programming experiment on which this thesis is based,

The categories of this coding schema could be described as a little vague, since several of the categories could be used in a number of quite different situations. As an example, "formulating strategy" describes a much more collaborative dialog section if used when both pair members contribute to the strategy and both understand it, than it does when used to describe a situation in which one pair member says everything, and the other is silent. The category could be used in both these situations.

6.4.3. Our Use of the Schema

Initially, this coding schema was considered as not so relevant for collaboration in pair programming, and the schema was therefore not a basis for any of the initial versions of the coding schema. However, the notion of mental models was introduced in the middle versions of the coding schema, since it was realized that they would describe an aspect collaboration that the other categories did not. These mental model categories remained a part of our schema, and were included in our final coding schema as well.

6.5. The Schema of Carol K. K. Chan

In the article "Peer collaboration and discourse patterns in learning from incompatible information", the authors' objective is to examine how big "the effects of peer collaboration on knowledge processing and conceptual change" [Chan 01] are, and how disagreements are handled. In the process of measuring this, they create a coding schema with a high-level focus. The schema divides sections of dialog into categories based on what type of collaboration is present. The categories are few; the schema consists of only four categories. Three of these are described as "surface moves". The first of these three is "Stonewalling – differences are rejected to minimize belief change". The other two are called "ignoring" and "patching/compromising". The three "surface moves" are regarded as less desirable for collaboration than the last category: "Problem-centred inquiry – differences are viewed as problems that need to be explained". Problem-centred inquiry is what the article argues should be strived for when trying to achieve successful collaborative learning. Through their experiment, the authors also show us that there was a quite large difference in the amount of use of the problem-centred inquiry category between groups that performed poorly and the groups that performed well. The best performing groups were those that used problem-centred inquiry the most.

Unlike in the other articles, the authors in this article state which of the categories that they consider to be "good" and "bad" for the quality of collaboration. These considerations could be useful to us in the process of making research hypotheses.

6.5.1. Advantages

This coding schema is not primarily intended for use in a pair programming setting or even for software development at all. However, the schema has certain qualities that make it noteworthy, and the schema is probably usable for coding nearly any kinds of collaboration-based tasks in which dialog is a central part.

The high-level focus of the schema might make the categories fairly easy to apply to the protocol. Also, the categories are defined in a way that makes them very separable from each other; more so for the categories in this schema than for the categories in most of the other schemas.

This schema seems to be quite suitable for our use if it is combined with some more specific categories that would make the final schema more capable of describing pair programming collaboration in its entirety.

6.5.2. Disadvantages

The schema seems a little simplistic and the categories are not very concrete. A major problem, regarding the schema's usefulness to us, is that the schema contains categories for describing conflicts only. The schema lacks categories suitable for describing situations such as questions being asked or explanations being made. In this article, as in the one above (described in Section 6.4), the focus is much more on learning than it is on problem solving.

6.5.3. Our Use of the Schema

Some of the categories from this coding schema, especially *stonewalling*, were helpful to us in describing the collaboration in regards to reactions to statements. The *stonewalling* category was included during the latest versions of the schema and is still there in our final version of the coding schema.

6.6. The Schema of Kathleen Hogan, Bonnie K. Nastasi and Michael Pressley

The goal of the study presented in this particular article, is to investigate the "discourse components, interaction patterns, and reasoning complexity" [Hogan 00] in groups of three people. The authors analyze the discourse by using three "interaction patterns" which are the following:

1. Consensual – Basically what is commonly known as "one-way communication", where one of the participants does most of the talking, and the other people merely agree or repeat what is said, or they might even ignore it.
2. Responsive – Several people contribute to the conversation.
3. Elaborative – Several people contribute to the conversation, and they do so in a way that elaborated on the previous contribution, so that the dialog was always evolving towards a solution for the problem. [Hogan 00]

The authors also use a collection of so-called microcodes. The microcodes are categories that describe actions at a different level than the "interaction patterns", and they are meant to be applied to single statements. The complete list of the microcodes is: *Presents (idea, partial idea, information, summary)*, *Repeats (self, other)*, *Elaborates (self, other)*, *Evaluates (own, other, task)*, *Reflects (standards, understanding)*, *Regulates action*, *Presents query*, *Requests information*, *Reacts (agrees, neutral)*, *Reacts (disagrees)*, *Digressions* and *Uncodable*.

A notable finding presented in this experiment is that during the experiment, there was a noticeable correlation between the subjects' achievement levels and their degree of participation on the discussions. More participation implied greater achievements.

6.6.1. Advantages

The interaction patterns mentioned here seem suitable for collaboration coding, and it would be possible to apply the categories in "real time" (as explained on Section 6.1 above). The "patterns" that make up the categories would lead to analysis material that would be interesting for us, since knowledge about the flow of the dialog seem like an important part of classifying collaboration.

6.6.2. Disadvantages

A small downside with the interaction patterns is that it will require a highly trained person in order to apply the categories correctly. There also seem to be some situations where the collaboration would be in the borderline between categories.

The "microcodes" seem to be insufficient for describing the complex area of collaboration.

The schema also shares some of the disadvantages from the article in Section 6.4. Both schemas focus more on learning and achieving "total agreement" than on solving a task.

Another downside is that the article discusses groups of three, and not pairs two (like in pair programming). However, the categories themselves seem to be equally suitable for coding discussions between only two people.

6.6.3. Our Use of the Schema

Elements of the schema were used with great success from the later versions of our coding schema and onwards. By using the interaction patterns from this schema, one could describe a very important part of collaboration, namely how the "pattern of interaction" was.

In situations where both programmers say about equally much during a pair programming session, they do not necessarily collaborate very well, for example if they never answer each other, but talk to themselves the whole time. When using the "interaction patterns" as categories, this kind of behavior is identifiable.

6.7. The Schema of Sallyann Freudenberg, Pablo Romero and Benedict du Boulay

This coding schema was presented in the article "'Talking the talk': Is intermediate-level conversation the key to the pair programming success story?" [Freudenberg 07]. The authors try to investigate whether the navigator and the driver usually follow the prescribed behaviors for the roles (where the two have a different level of abstraction) or if the two programmers are focusing on the same things instead. The article lists some interesting findings based on four one-week studies of commercial programmers. The conclusion is that the driver and the navigator typically make the same amount of utterances in the different abstraction categories.

Not surprisingly, considering the focus of the article, the coding schema focuses on abstraction levels. The categories are few and well-defined, and are shown in Table 4 below.

Code	Explanation	Examples
SY	Syntax – Spelling or grammar of the program. Spelling is indicated in the transcriptions by single letter capitals. NOT semantics.	S P E L L I N G, dot, F9, 7.
D	Detailed – refers to the operations and variables in the program. A method, attribute or object which may or may not be referred to by name.	This condition, that return value, the list, the counter, what this returns or gives, getCustomer.
PR	Blocks of the program. Including tests and abstract coding concepts. Also strategy relating to the program and its structure. General naming standards discussions etc. This could also include cases where the subject of the sentence refers to 'some of them' or 'they all' – i.e. a group of conditions. Anything to do with refactoring. Subsystems or libraries. Directories or paths, even if named.	That loop, truncation, the error handling, Oracle, this issue. this part of the program, mock, Mosaic.
BR	The statement bridges or jumps between the real world or problem domain and the programming domain. This may be where a case or condition exists in the code and the real world.	So we need to add a test condition here, to see if the bank account is valid for this kind of transaction.
RW	Real world or problem domain	Savings account.
V	Vague, including metacognitive statements and questions about progress or understanding. References to a place on the screen. References to the development environment and/or navigating its menu structure.	Oh, yeah, I see, that bit at the top.

Table 4: Freudenberg et al.'s coding schema.

6.7.1. Advantages

The coding schema presented in this article, focuses on mental models. This was also the case in the articles described in Section 6.3 and 6.4 above. The categories in this schema, with focus on the level of abstraction among the developers, might provide interesting information about the collaboration. Even though the programmers seemingly cooperate very well, if both contribute to the collaboration with suggestions and feedbacks, and both argue well for their ideas, the actual cooperation might not be very beneficial if the programmers discuss things at one end of the abstraction level range only. Constant arguments on spelling or coding standard does not automatically lead to good software (even though the code might look nice), and prolonged philosophical discussions about the intended goal of the software does not help much if the code does not compile. As the article suggests, a balance between the two extremes is best.

The categories are well defined, with understandable examples, and will be fairly easy to apply to the verbal protocol in near real time. Because of the rather low number of categories, the task of choosing between them on ambiguous statements (where "vague" is not an appropriate category) will be easier than otherwise. Moreover, the schema was developed for use in a pair programming setting.

6.7.2. Disadvantages

While the categories are few, especially compared to the coding schemas in other articles, one might still wonder if they are few enough. When applying the categories to the protocol, there will likely be some situations where it will be hard to choose between the D (Detailed) and the PR (Blocks of program) categories. These categories appear as quite different when reading their definitions and looking at the examples in the article, but how large a section of the code must something be before it is called "a block" of the program? This is not explained in the article, and could be interpreted as anything ranging from a line of code to everything that is larger, even though they probably mean (since they mention a loop as the "smallest" example of this) that the block should be at least a couple of lines.

6.7.3. Our Use of the Schema

From the middle versions of our code schema, and continuing even into the final version, the spirit of this schema was included as an important part of our coding schema. Initially, it was included as one of two subcategories to be applied; in later versions – it was one of six. To remove the disadvantage described above, the "PR" category was not included in our schema, so that it would be easier to code a low cognitive level. Also, "D" and "BR" was combined, and "V" was renamed and redefined in order to include only Metacognitive statements. The concept and definition of "Metacognitive" was improved and inspired in part by its description in the article described in Section 6.6 above.

6.8. The Schema of by Sallyann Bryant

This schema appears in the article "Double trouble: Mixing qualitative and quantitative methods in the study of eXtreme Programmers" [Bryant 04]. In this article, the author's objective is to study pair programming in general, and specifically look for behavior differences in pairs depending on the pair members' level of experience with pair programming. Bryant presents the interesting finding that people from pairs where both have much experience with pair programming act more similar to their partner than people in the low experience pairs do. In these less experienced pairs, people act much more according to the pair programming role they are in (driver/navigator). The highly experience pairs also review and test the code more often, and they make explanations and suggestions less often.

The categories presented in this article are pretty straightforward, and similar to those in previously mentioned schemas (especially the ones in Sections 6.1 and 6.2). The total list of categories is the following: *Question*, *Explain*, *Suggest/counter*, *Confirm/agree*, *Remind*, *Change driver, Look up information*, *Review/refactor*, *Test*, *Rest* and *Other*.

The categories are not that well explained; about half of them do not have any explanation at all, and the explanation for the rest is only one sentence without any examples.

Its advantages and disadvantages will not be discussed further, as they will be the same as for the two schemas this one is similar to. However, the few and loosely defined categories would ultimately not be a substantial influence for the final versions of our coding schema.

6.8.1. Our Use of the Schema

This schema was a good basis for the very initial versions of our schema, since it was straightforward and easy to grasp. As mentioned, the article presented some interesting findings, which led to the hope of finding equally interesting results in our study if the same schema was used.

6.9. The Schema of Gary M. Olson, Judith S. Olson, Mark R. and Marianne Storrøsten

This schema is from the article "Small Group Design Meetings: An Analysis of Collaboration" [Olson 92]. The article is based on CSCW (Computer-Supported Cooperative Work) ideas, and the development of the coding schema in it is the goal of the study. The authors observed a number of groups consisting of 3-7 people when they were having meetings about design. It is stated that about 50% of the time that was spent in the design meetings did not concern new design ideas, but consisted of the people doing administrative duties like coordinating the work and summing up the work already done.

The coding schema itself is simple and not universally applicable. The full list of categories is: *Issue*, *Alternative*, *Criterion*, *Project Management*, *Meeting Management*, *Summary*, *Clarification*, *Digression*, *Goal*, *Walkthrough* and *Other*. As one can clearly see, the focus is on meetings primarily, and not on pair programming at all. This is understandable, since meetings were the focus of the study in the article.

6.9.1. Advantages

All categories are well-defined. Since they were developed based on observation of the meetings, they are probably very good for design meeting analysis. The categories also seem simple to apply, and the borders between them are clear. Coders would rarely need to think long about which category to use.

Some of the categories are interesting since they focus on a slightly different area than the other schemas do. The categories such as "issue", "criterion" (arguments, opinions) and "goal" makes it seem like this schema is a little more systematic and result-based than many of the other schemas reviewed in this thesis.

6.9.2. Disadvantages

Many of the categories are quite irrelevant for describing pair programming; especially the meeting-based categories (project management, meeting management). The categories are also quite few. Additionally, the categories, while well-defined and disjoint, seem quite vague. Another slight disadvantage for its usefulness in our study is that the groups for which this schema is developed are 3-7 people in size, and not pairs.

6.9.3. Our Use of the Schema

Since the categories are simple and varied, inspirations were taken from them for some of the early versions of our schema.

6.10 The Schema of Takeshi Okada & Herbert A. Simon

This article, named “Collaborative Discovery in a Scientific Domain” [Okada 97] is a mainly psychology-based elaboration. The authors present an experiment in which the goal was to investigate differences in performance between pairs and individuals in what they call a “discovery problem”, which was a small theoretical biochemical assignment. They note that the pairs performed significantly better than the individuals. They explain this finding both by mentioning seemingly obvious facts (like that the pairs are two, and thus have a double chance of getting the answer right), and deeper explanations: The pairs were much more likely to change their hypotheses and justify their choices than the individuals, who were thinking aloud, were.

To measure the behavior of the subjects, the authors use a coding schema that is made for describing construction and adjustments of hypotheses.

The advantages and disadvantages for this schema will be the same as for the high-level schemas discussed earlier in this chapter, and will not be discussed in detail. However, this schema’s focus on creation and explanation of hypotheses (like “Agreement to the hypothesis”), rather than focusing strictly on what is done (for example “answering question”), is interesting. This schema is not only measuring what is done during the collaboration, but also what kind of, and how big an, impact the collaboration has on the decision making and final result.

A hypothesis agreed on might not always be correct, for example, in the case where the contributor of an erroneous hypothesis is a good negotiator/persuader and successfully persuades his partner into accepting his idea. However, we can be certain that the hypotheses agreed on by two people are generally more thought through than the hypotheses individuals make, since the individuals might base their choices on gut-feeling or similar unchecked apparatuses of decision without upsetting a peer. If one of the two in a pair attempted something similar, the other member would likely protest to the hypotheses or at least request a better explanation for it.

6.10.1. Our Use of the Schema

Although not plainly visible, inspirations from this schema were of good use for the “Begin Characteristics” group of subcategories and, to a lesser degree, “Interaction Patterns” and “Result”, in our final schema versions.

7. The Development of Our Coding Schema

Our coding schema was revised significantly six times before it was decided that the schema was sufficient for our needs, and the actual coding process could begin. In addition to these seven more or less discrete coding schema versions, there were continuous minor adjustments to the categories' definitions throughout the process of developing and testing the schemas. (The initial version of our schema consisted of two parallel schemas, so eight schemas actually existed in total.)

The order of development and how the versions influenced each other is illustrated in Figure 2. The arrows indicate the directionality of development and influence, and points to the schema version that was influenced. Bold headlines signify the major schema versions. The major versions are regarded as "major" if they are the result of large changes to the previous version. All iterations are described in detail in the following chapters.

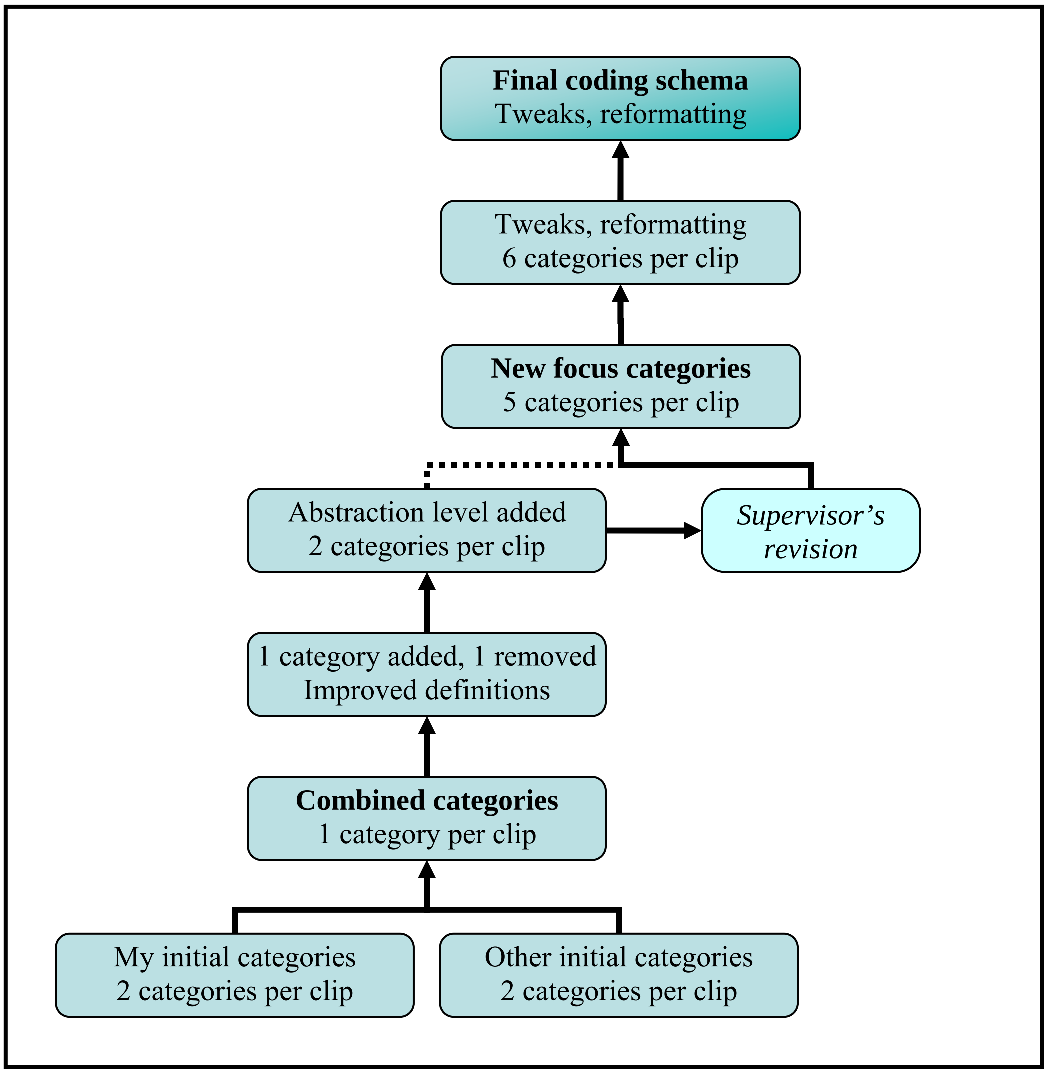

Figure 2: Coding schema development

7.1. Initial Category Draft.

7.1.1. My Version

After reading through relevant literature, the other master student and I, made one individual coding schema each, and tried to apply the categories from each of our schemas to one audio file.

My initial schema focused mainly on what the pair programmers said, and how the programmers reacted to statements made by the other person. The schema was mainly influenced by the schemas of Cao and Xu (Section 6.1), Bryant (Section 6.8), Bryant et al. (Section 6.2) and parts of the schema of Hogan et al. (Section 6.6). My first schema is shown in Figure 3 below.

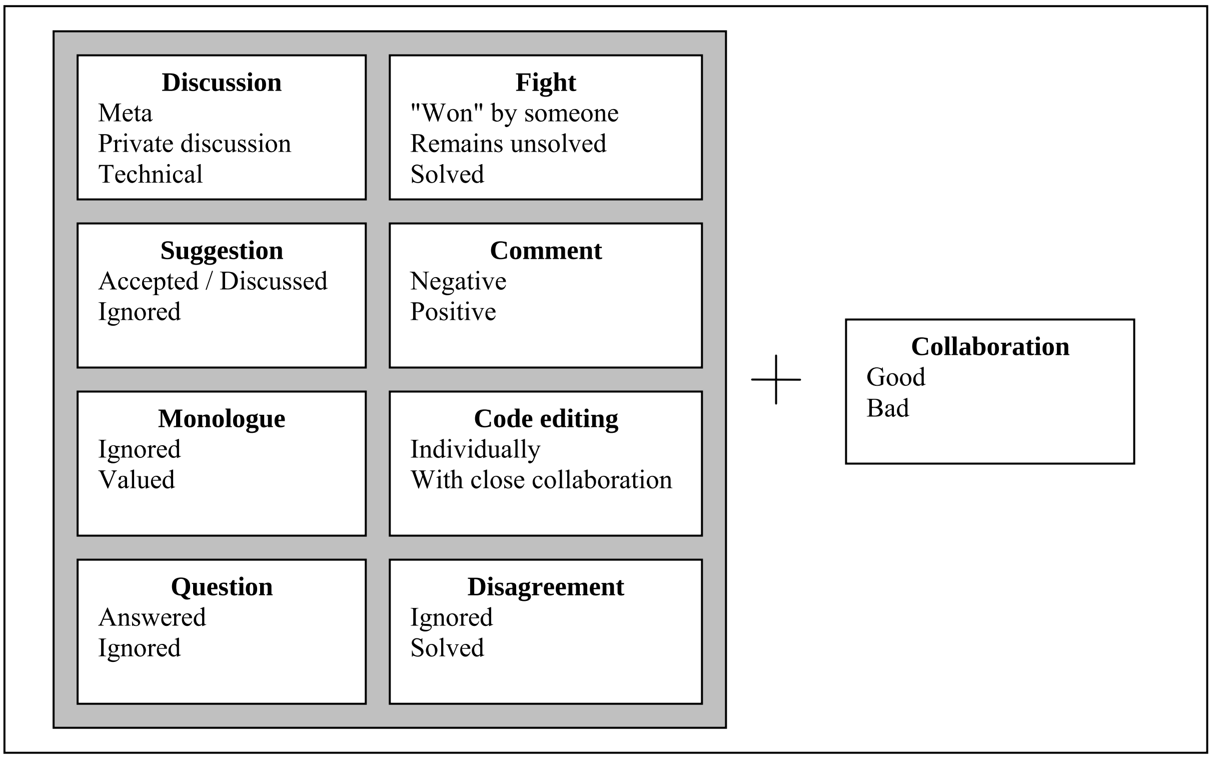

Figure 3: My initial category draft.

The category *Meta* described discussion related to, but not directly about the task. *Technical* described discussion about solving the programming task.

At this point, we were inexperienced in use of the Transana software. In this first schema, the categories were ordered in a hierarchical manner. The planned coding procedure would consist of assigning the clip to an activity-describing category chosen among all the ones in the gray box in figure 3 above. For example, if one of the programmers asked a question, and the other one answered it, the category Question(Answered) would be chosen.

In addition to the activity category the clip was assigned to, in order to describe the activities that were being performed by the pair, each clip was meant to be assigned to a category describing the quality of collaboration. This qualitative characterization consisted simply of "*good*" or "*bad*". Accordingly, every clip would be given two categories in total; one category describing the activities performed by the pair programmers, the other category judging whether the collaboration was good or bad.

The audio file I coded with this schema had a total duration of two hours and twenty-one minutes. I divided it into 81 clips (averaging 1 minute and 44 seconds in length). Of the total selection of 20 subcategories, I used 15 of them, but I used only eight of them more than twice. The clips that I coded as *code editing* were by far the longest in average. Sometimes they were as long as five minutes or more.

7.1.1. The Strengths and the Weaknesses of the Schema

My categories turned out to be less than ideal. Many of them were superfluous, and the separate rating of good/bad collaboration was not a great idea. For many of the activity categories, a certain collaboration category always followed. For example, the activity category *editing(cooperatively)* was only used for clips that I coded as *collaboration(good)*. For the clips I coded as *monolog(ignored)*, I always used the collaboration quality category of *collaboration(bad)*.

Because of my imperfect coding schema and my low level of experience with coding at this point, many clips were not assigned to exactly two categories (activity + collaboration quality). Several clips were coded as both good and bad collaboration, for example, in situations where the activity remained the same throughout the clip, but the collaboration quality level fluctuated.

I had not included any specific guidelines for the length of the clips or for the use of the categories, and the length of my clips turned out to vary a lot. For example, clips coded as “programming” would sometimes last for several minutes, while clips of question(answered) only lasted for a couple of seconds. In some situations, I used two activity categories for one clip, since a short question inside a long programming session was considered to be too insignificant to require a separate clip at this time. This was in retrospect not a good practice, since the categories which only described parts of the clip would then be given “full credit” for the clip, and the time the category was used would increase by the whole clip length. Since these “insignificant” areas were often much shorter than the clips, the statistics generated about time use of categories would be incorrect.

Another problem with this initial schema was the very high level of subjectivity involved in the application of *collaboration(good/bad)*. This problem was identified early, and there were fears that it was tempting to be too “kind” and give the programmers more “good” collaboration than they had deserved. At the time, it was thought that the collaboration category could be applied with more objectivity after becoming more familiar with the material and the categories after trying to code some more audio files.

The practice of judging manually whether the collaboration during the discussion in each clip was good or bad was abandoned in the next schema suggestion. However, the concept of indicating whether or not the collaboration was good or bad remained a part of our schema for several versions, although it was from the next schema version on built into the categories, and did not require a subjective judgment when coding. At even later schema versions, this measure of good/bad collaboration was eventually removed altogether.

Although my first schema was imperfect, it did have its strengths. Large elements of the schema were based heavily on previously published schemas. Also, a notable amount of my initial schema has been kept through the next versions of our coding schema; some of my categories were kept until the very end. Another advantage with this schema was that is was easy and fast to use. When I made clips and allocated them to categories while listening through the file, I used only 40% more time than the duration of the coded section. The schema was based on some good ideas, but its actual usage was not sufficiently thought through and the schema lacked a lot when it comes to grasping the "bigger picture" of collaboration.

During my coding work on the first audio file, lessons were learned about the difficulty of starting and ending clips at appropriate times. At this time, we had not made any specific rules or guidelines for clip length, but clips of about one minute in length were considered reasonable. Compared to the desired clip length associated with the more current versions of the schema, even the short clips in this coding attempt are actually quite long, (and the long ones here are incredibly long by our current rules.)

7.1.1.2. Extra Information added to the Coding

When coding with this schema version, I would write short comments about what happened in the text area of Transana while making clips and applying categories. This was a good routine when using my first schema, since the routine worked as a way of briefly describing why a particular category was chosen. Sometimes, however, I could not write anything else than the category name here, since it was hard to describe the contents of the clip in other words than to use the name of the chosen category.

Additionally, at the top of the transcription text area of Transana, I wrote a really short description of the two programmers. I gave them the names *P1* and *P2* and wrote just enough about them to distinguish them from each other. For example, I would note the gender or the dialect or accent used if the two pair members varied on one of those factors. I also attempted to note which of the two the driver was and who the navigator was when they started the programming tasks.

At the bottom of the transcription text area, I wrote a short and subjective text of 2-4 short paragraphs which described my overall impression of the collaboration. I also mentioned the development of the collaboration as time passed and how, and if, the collaboration changed when the pair members changed pair programming roles.

7.1.2. My Fellow Master Student's Version

The second of the two individual attempts to make an initial coding schema was done by another master student. Her schema turned out to be quite similar to mine. Because much of the literature I read was read by both of us, and because she and I had a number of discussions before we started developing a schema, the similarity between our schemas was not unexpected.

In her schema, as in mine; the notion of "good" and "bad" collaboration was used. However, unlike in my schema, good/bad were in her schema meant to be supercategories, with all other categories being subcategories of these.

The subcategories in her schema, which were similar to the supercategories in the schema described in Section 7.1.1 above, were few in numbers, but they had a rather deep focus. The concept of "solution" was used. Not only was the reaction and behavior of the two programmers significant here, but also whether or not suggestions made by one of the pair programmers lead to a positive result. For example, a suggestion can be accepted by the other person, but lead to a course of action that turns out to be wrong. An other suggestion that is controversial and disputed could turn out to lead to a correct result.

This schema, like mine, was meant to be used along with some subjective notes at the end. A description of how the change of roles mattered, how the collaboration developed over time, and the apparent personality of the two programmers were stated as things worth listening for.

7.2. Combined Categories

7.2.1. Development

After we had developed the initial schemas, we had a meeting with the intention of merging our two schemas or developing a whole new schema based on them. We chose to make a new schema, since our two individual schemas would not merge well due to the large structural differences between them. However many elements from our initial schemas were included into the new one. From my schema, we kept most of the actual categories. From my fellow master student's schema, we kept the structure, and the notion of having good and bad collaboration as supercategories.

Inspiration was gathered from several articles as well, and we increased the amount of categories based on new ideas that we found in the literature. In addition to the articles used as our basis in the first phase of the schema development, ideas from Olson, et al. (Section 6.9) Lim et al. (Section 6.4) and Chan (Section 6.5) was included. Our combined schema, the second in the line of coding schema versions, is shown in Figure 4 below. The category definitions are listed in Appendix X1.1.

Bad Collaboration	**Good Collaboration**	**Other**
One-sided break	Discussion with agreement	Ad-hoc-work
Suggestion ignored	Suggestion accepted	Break
”Override”	Planning of future work	Private discussion
Passive person	Programming, duo	Unsolved disagreement
Programming, solo	Question answered	
Question ignored	Disagreement solved	
Outside work		

Figure 4: The combined category schema

In this new schema, the structure implied that a certain collaboration rating (good or bad) was automatically applied depending on the activity category used. For example, “Suggestion that is ignored” would be categorized as bad collaboration and “Question which is answered” would receive the opposite collaboration grade, namely good. In the hierarchic ordering of the categories, four subcategories were subcategories of the collaboration-neutral supercategory “other”. These “other” categories were meant that situations where neither a classification of good nor bad collaboration would be fully appropriate. Pauses and off-topic conversation were among these.

Six of the subcategories we included in our schema were placed as subcategory of “good”, meaning that we regarded them as beneficial for the collaboration. Seven categories were regarded as “bad” for the collaboration. For this schema, all categories were described and/or exemplified.

7.2.2. Usage

After some discussion about how we would test this new schema, we decided that it would be best if we coded the same parts of the same audio files in order to check whether or not the categories were sufficiently objective and well-defined that we would give the same areas of dialog the same coding. We also discussed which files and how much of them we would code in this test. We decided that we would code parts of

- two files, in order to reduce the risk of choosing an unrepresentative file to test the schema on.
- two different areas of the files, in order to avoid coding, for instance, the start of two files.

The final decision ended on coding thirty minutes of two different files. For one of the files, we began coding from the beginning of the file; for the other one, we starting at the forty minute marker and coded thirty minutes of the file from there.

When we each had coded the agreed file sections, we had a meeting in which we checked whether or not our agreement on the used categories was good. Agreement turned out to be more than acceptably high when we counted our use of the supercategories, “good” or “bad” cooperation only. However, our agreement was not good at all when taking in account also the subcategories. If we included the start and end time marker placement of the clips as parts of the agreement calculation, we were almost never in agreement.

Immediately thereafter, we had a discussion about the categories that were used most often on clips where we disagreed on the coding. This discussion solved a couple of misunderstandings and conflicting conceptions we had regarding the categories. We exchanged some experiences and made some more specific rules about where to place the start and end marker of the clips. Clips would start where the new activity started, and end where the next activity began. No part of the file would be without clips and categories. At our meeting, we also made the important and final decision to use only one category per clip. This had been the informal consensus ever since we made this combined schema, but not until at this point did we include the “one clip rule” explicitly in the schema description.

We agreed to code 45 minutes of another audio file, this time thinking about potentially unneeded categories or categories that had problems with their definitions.

7.2.3. Category Quality

Thus, this second coding schema turned out to be a clear improvement from our individual first attempts described in Section 7.1. The new and more thought-through structure of the category hierarchy, where the collaboration quality categories of *good* and *bad* were above the rest, was perhaps the most important improvement at this point.

Since the main thought and idea of our new coding schema was pretty much the same as in our individual categorization attempts, the categories experienced the same advantages and disadvantages as those (as described in Section 7.1.1). In short; the categories were easy and quick to apply, but they suffered from being subjective, and there were still some superfluous and poorly defined categories in the mix.

The decision to use only one category per clip was an important one, and it made the task of making clips less subjective, since it forced us to make more, and therefore shorter, clips. Regrettably, there was no noticeable increase in correspondence between the clips the two of us made for the same time areas compared to before the "one clip" rule was introduced. Also, the clips remained quite long, averaging about one minute, but clips lasting for up to four minutes were not uncommon. Some categories, such as *planning of future work* and *pair programming* were the main "culprits"; clips coded as one of these categories were often long ones. *Question answered* and the other more specific categories were mostly applied to shorter clips.

We kept the practice of writing an overall evaluation of the collaboration in the audio file at the bottom of the transcription area, plus a short description of the two programmers at the top of it. We understood that this was an extremely subjective practice, and that it would most likely not be useful for our analyses. However, these notes were considered a useful tool for checking that we, at least, had the same overall opinions of the pairs in the audio files that we coded. To write these short descriptions took only a couple of minutes, and were thus considered to be worth the effort.

7.2.4. The Removal of the Least "Polite" Categories

At this point, we realized that the categories describing situations in which the programmers behaved least politely towards each other (such as "feud not resolved", "disagreement ignored" and "aggressive persuasion") were those categories that we had most frequently removed from our coding schema this far. During our coding schema revision meetings, we observed that the least polite categories of all were never used. Obviously, that the programming pairs were much more polite towards each other than we initially believed. The rather impolite categories were used extremely rarely, and often when they were used, the impolite behavior or statement was not necessarily meant to be impolite, but could just as well have been a joke. Though not altogether absent, even mildly subconsciously impolite situations such as those that would be coded as *interruption* or *suggestion rejected* were few and far between. This low use of the least polite categories would prove to be observable also for the later coding schemas.

7.3. Improved Categories

7.3.1. Meeting

Later, we had another meeting in which we compared our results when using the new code schema described in section 7.2. When counted informally, the level of agreement was encouragingly high; 90%. The rules for this informal counting was that it was regarded as "agreement" on a clip if the category we had used matched the other's on the "good"/"bad" scale, and that the time slot of the clip was fairly similar. However, a much lower agreement score much lower was found when considering the subcategories, and when we were stricter in regards to the start and end time of the clips.

A couple of major decisions were made at this point. The time the programmers spent filling out the forms at the start and end of tasks would be excluded from the analysis, and thus also not be coded. Also, the long time sink task (task 5), which we had in previous coding tests coded, would also be omitted. This task often took half of the total time for all tasks, so the decision to omit this task reduced the time it took to code a file considerably.

The concept of "good" and "bad" and whether or not to have these subjective supercategories was discussed a bit at this point, but no final decisions about it were made yet. The supervisor suggested at this point that the coding of collaboration might be better if based on cognitive aspects rather than a subjective measure of whether or not it was "good". A first attempt to cover these cognitive aspects was included in the schema at this point. This attempt proved to miss the mark a bit, so in the next category revision, it was replaced with a new system for assessing the cognitive aspects.

7.3.2. Tweaks

The general idea of our second category revision, producing the third version, was to make sure that the categories were disjoint and that they had clearly defined boundaries, so that there was no question about which one to choose in a given situation. Many of the changes consisted of nothing more than a redefining of existing categories. Those changed were on categories which we had previously defined rather vaguely and those where the other master student and I had a slightly different view on when to use them. If we wrote down an exact definition, our understanding of the category would be more similar. If we both were careful to follow the new definitions when coding in the future, our coding results would likely be more similar to each other's.

The category named "break" was changed. It now indicated that both participants took a break, and that at least one of the two was physically away from the computer. Earlier, the "break" category had no specific definition, since we thought the meaning of it would be self-evident. It turned out that the category was not that obvious, and it was sometimes used on clips where "private discussion", "off task work" or "one-sided break" would be more suitable categories to use.

The category "Ad-hoc work" was removed, since it was a category for which there was rarely a situation where it seemed suitable to use it. "Discussion with agreement" was another category we considered to be superfluous at this point. Although there was nothing wrong with the category itself, it was seldom used, and when it was used, other categories could just as well have been chosen instead. The same applied to "disagreement ignored", so that category was removed as well. Its definition was blurry and the category seemed to overlap "disagreement not resolved" and to a lesser degree; "ignored suggestion".

The overly used category "Planning of future work" was altered in an attempt to reduce its use, which was out of control. The initial definition of the category was broad, so it was used a lot. In some particularly imperfectly coded areas of, the category would be used on nearly all clips, except for those which were not comparable, such as breaks. A requirement was added to the definition of the category: from now on, both programmers needed to contribute substantially to the discussion; the ideal was a contribution of 50% to the discussion from each programmer, but values as extreme as 80/20 % were tolerated.

"Private discussion" was another category that was poorly defined in our coding schema. The new definition for this category aimed at removing the potential for misunderstandings in situations where the discussion was not directly task-related, but where the discussion could not be said to be of a private character either. The new definition specified that what was said had to have absolutely no relevance to the task at hand. An utterance such as "do you have much experience with Java?", would be seen as "planning of future work" or something else, since it was task relevant, since knowledge of the other person's expertise level could influence how the dialog would develop.

The “solo programming” category was slightly altered in definition to specify that at least one of the two programmers were passive in clips where the category was used, and that there was actual programming taking place during it.

One new category was introduced and named “other tasks”. This new category was meant to be used for describing small tasks that were related to the experiment, but which did not involve any relevant collaboration. Picking up printouts and talking to the experimenter about the rules of the experiment or other things (which actually happened quite often in the early parts of the audio files) would be examples of this.

The new category schema is shown in Figure 5 below. The category definitions are listed in Appendix X1.2

Bad Collaboration	**Good Collaboration**	**Other**
One-sided break	Suggestion accepted	Other work
Suggestion ignored	Planning of future work	Break
”Override”	Programming, duo	Private discussion
Passive person	Question answered	Unsolved disagreement
Programming, solo	Disagreement solved	
Question ignored		
Outside work		

Figure 5: The improved combined category schema

7.3.3. New Ideas

As mentioned above, in order to fulfill the need for more focus on the cognitive dimensions, a new level of categories were proposed. The categories from this new level of categories were not meant to describe clips, which were rather described by our usual activity-based categories. The idea was that a coder would make a group consisting of several already categorized clips, and define a large clip based on this group, and then give this large clip a cognitive category.

These cognitive categories were based on the evaluation questions the programmers’ themselves were meant to fill out after programming was complete. The questions included how thoroughly the programmers read the task description and understood it before beginning, and how easy they thought the task was to solve. For our cognitive aspect categories, we would evaluate the programmers on these factors.

The additional set of categories was ordered like the activity-based categories, with “good problem solving” and “bad problem solving” at the top level. Under these supercategories, there would be a likert scale-based selection of answers to the following three questions. Which were the three questions among those in the self-evaluation form that we thought would best describe the collaboration.

- Did they (the programmers) have a clear understanding of how the task would be solved before they started coding?
- Did they discuss the task thoroughly before starting to code?
- How was their strategy for solving the task?

The notion of "mental models" was mentioned in our meetings, but it was not included in the coding schema at this point. It was considered to be too hard to extract from the verbal protocol whether or not a pair or an individual had a solid mental model and how this model evolved during the course of the discussion. The programmers were, from what we had heard during listening through and coding the audio files, quite goal-oriented, and not as interested in understanding the big picture of the task. Also, it was feared that some people that said little, might have a really comprehensive mental model of the task and the problem space, but did not talk about it. How one could know if what people said really represented their mental models, was a question of concern at this point.

7.3.4. Usage

Our new and/or improved categories were tested on one file. When we compared our coding of the file using the same tolerant standards as before for counting "agreement", the agreement was, as previously, very high.

A problem occurred when we realized that Transana's category system does not support the kind of ratio scale categories that we planned to use when evaluating the cognitive dimensions. Due to this and other factors (most importantly that we realized that our suggestion for measuring the cognitive dimensions was a little pointless and that it was not documented in the literature,) our cognitive attempt was discarded merely days later when the next suggestion for the schema was developed.

7.4. Expanded Categories

7.4.1. The Table

At this point, we had achieved a basic familiarity with different coding schemas and the pair programming audio files. In order to increase the quality of the categories, a real "wash" of them was requested, where we should try to make sure that most categories had some backing in the literature. To help in finding similarities between our schema and those presented in the available literature, a large table was made, consisting of the most relevant articles we had read this far. The table was ordered so that our own coding schema could be compared to the others easily: A category from one schema was placed on the same table row as a similar category from another schema. Though the table showed that a majority of schemas focused on categories similar to those we already had in our schema, a couple of very interesting categories we found in the table had no match whatsoever with any of the categories in our schema. Also, a few of our categories in our schema had no matches in the literature either.

7.4.2. New Suggestion

Based on what we learned when making and looking at the table described above, a range of new categories, meant to describe the cognitive aspects of collaboration were made. (As mentioned above, our first attempt at making cognitive categories was discarded.) Our new cognitive categories were heavily based on the coding schema in the article by Freudenberg et al. (Section 6.7 above.) The resulting coding schema, the fourth schema, is shown in Figure 6 below, and its categories are defined in Appendix X1.3.

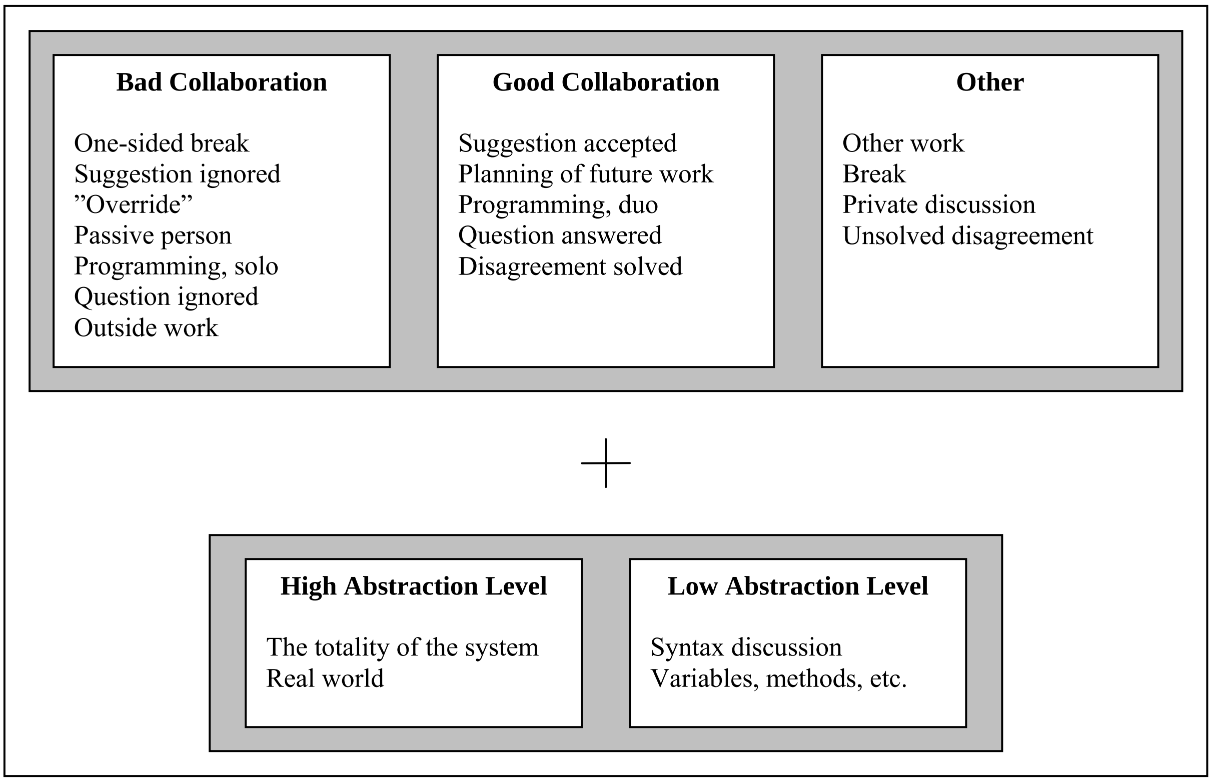

Figure 6: The expanded category schema

Our new categories would be ordered in two levels, since this amount of levels is what Transana encourages. The upper level would consist of the supercategories *high abstraction level* and *low abstraction level*, each having a couple of subcategories. To have a high abstraction level meant to be focusing on the big picture of the system, objects and relations and references to the real world counterpart of the system (in the case of the pair programming experiment: a real coffee machine). A low abstraction level would imply focus on the code itself. Syntax, loops and spelling would be obvious examples of this. Methods/functions of the code would be the most abstract of concepts that would be suitable to identify as belonging to the low abstraction levels.

A third top-level category was also made. It was named *NA/other* and it would be used when the discussion was not about the code or programming at all, or when the discussion was all over the place and back and forth towards the two more specific cognitive levels with very short intervals and one could not make sense of it.

In addition to being based on the schemas in the available literature, the new cognitive categories were inspired by Shneiderman’s book “Software Psychology” [Shneiderman 80], where he states.

> The syntactic/semantic model suggests that programming-related knowledge is split into two domains. Semantic knowledge, the first domain, is meaningfully acquired, language independent, resistant to forgetting and hierarchically organized from high-level problem domain related issues to lower level functions of programming, Syntactic knowledge, the second domain, is acquired by rote, language-dependent, easily forgotten if not used, and arbitrary.

Each clip was now meant to have two categories assigned to them. One category would be selected from the collection of categories that we had been working on from the beginning, i.e., the activity-based ones such as “question” or “programming”. The second category would be chosen among the new categories: the cognitive ones. All clips would have exactly two categories, and now that the *NA* (not applicable) category was introduced for the mental model focus; that would always be possible, since one could choose *NA* if no specific cognitive level seemed appropriate.

7.4.3. Individuals or Pairs?

At this point, it was heavily discussed whether the cognitive aspect categories should be assessing the cognitive level of the pairs as a whole, or if they should rather be measuring the two pair members individually. We decided that it was best that pairs were the focus, with regards to both the cognitive- and the activity-based categories. One could ignore individual differences between the two programmers. For example, it was not important who interrupted the other person, but it was important that interruptions occurred.

If the opposite had been decided; that individual behavior was to be recorded, the assignment to the categories would have taken much longer. A coder would then be forced to always distinguish between the two people, and note for every clip who the initiator and who the responder of the particular interaction sequence was. The individual focus would also be awkward to perform in Transana, where this kind of focus would require a lot of repetitive micromanagement-like actions when applying the categories.

7.4.4. Usage

The short time frame from this revision to the next one led to that the schema was not tested on any complete files. Thus, results from using the schema were not available for evaluating the schema. Only smaller test were made in order to gain familiarity with the new categories.

7.5. New Focus Categories

7.5.1. Comparison to the Previous Ones

Shortly after the schema version described in Section 7.4 was completed, we realized that we were not so limited by the two-level category system of Transana that we thought. We now had new possibilities regarding ordering and the hierarchy of the categories. As well, further studies of the large table that compared coding schemas, made us realize that many interesting factors of collaboration were not covered by the categories in our schema.

Based on our coding scheme from Section 7.4 and the findings in the large schema table, the supervisor made a completely overhauled, fresh-looking schema suggestion. This new schema had several parallel collaboration foci, which was now possible, since we had realized that Transana did not limit the complexity of the coding schemas if we used the search functionality of Transana as well as the regular analysis functionality when summarizing the coding results.

The schema is shown below, in Figure 7 below.

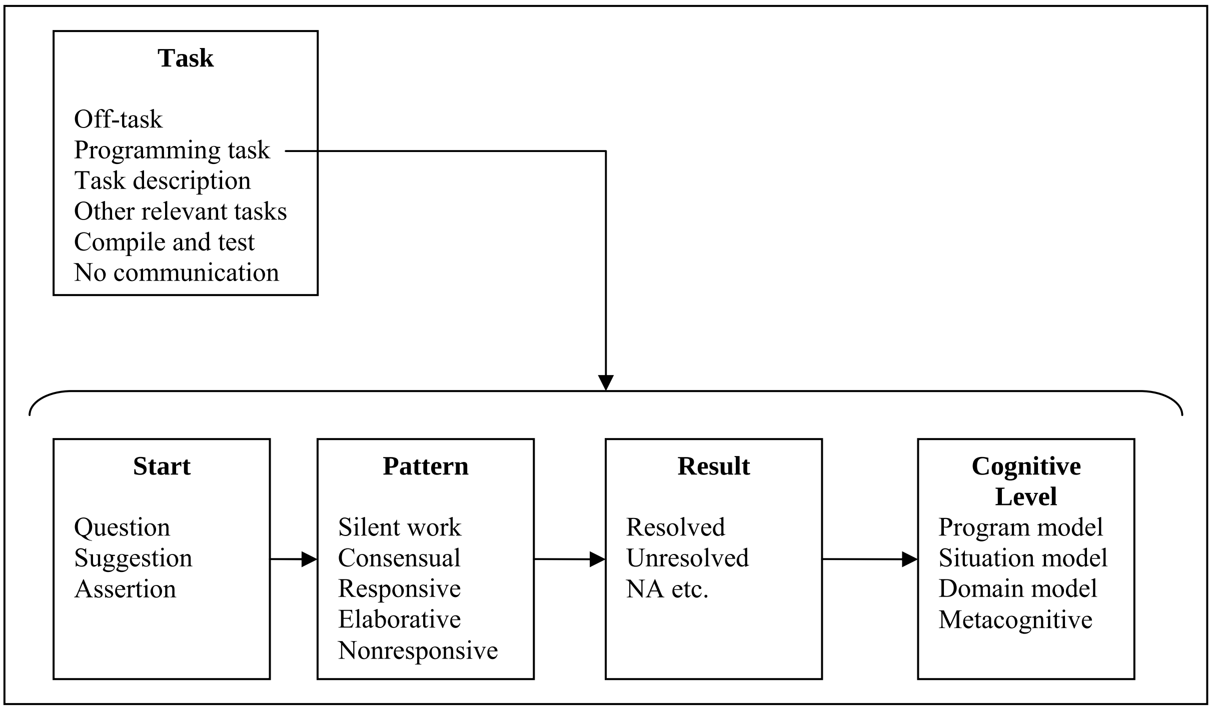

Figure 7: The new focus category schema

The new schema was seemingly very different from our earlier versions, but was at its core quite similar to them. The major difference was that the new schema version consisted of categories that were more atomic than before. The categories described only a small part of the contents of the clips in this new schema, for example that it contained a question. In our earlier schemas, the categories described the full clip alone, for example that the clip consisted of a question being asked and answered sufficiently.

We thought and hoped that the more atomic categories of the new schema would be less subjective than our earlier categories. However, one had to assign a clip to a larger number of categories than before in order to completely describe the activities of the clip. In the new schema most clips were meant to be assigned to five categories (compared to the two, as it was in the earlier schemas), in order to describe the content of the clips. These five categories were supposed to be chosen from the following five category groups below. One subcategory from each group:

1. Task: What the pair programmers are doing. Examples of task categories are *programming*, *discussing task description*, *testing*, and *off-task conversation*. Only if the task is *programming*, will the coder assign the clip to the remaining four category groups. Clips consisting of dialog coded as, for example *testing* will be assigned to a task category only, and nothing more.

(The three middle categories collectively describe an *Interaction pattern*: How the dialog flows)

2. Start: What sort of utterance initiates the dialog? *Question*, *suggestion* or *assertion*?
3. Pattern: How is the reaction to the initiating statement? Is it a good and full response that contributes substantially to the dialog (*responsive*), or just confirmation or an mm-hmm (*consensual*). *Nonresponsive*, *elaborative* and *silent work* is the other choices.

4. Result: How is the situation at the end of the dialog pattern compared to before? Was the problem which was brought up at the start of the interaction pattern resolved or not?

5. Cognitive level - a renamed version of "*abstraction level*" that we implemented in our fourth schema version (Section 7.4). It remained largely unchanged, but the number of subcategories here was decreased slightly to make sure they were disjoint.

The categories of our previous schemas would fit into several of the new category groups. Some of our old categories were kept, and placed in the *task* group, and most of our old abstraction level" categories were transformed into the Cognitive Level group of the new schema. However, most of our previous categories were less specific than the categories of the new schema, so a clip we earlier coded with the single category "question answered" would now require both the *start* category "question" and the *pattern* category "responsive". The advantage with this more atomic system was that it would be more versatile than the more compounded categories we had in our previous schema versions.

A major difference between this new schema and our older versions, other than the structure rethinking, was that categories similar to the macrocodes in Hogan et al. (Section 6.6) and the categories of Chan (Section 6.5) were introduced. These categories would in our new scheme, constitute the category group *pattern*. In our earlier schemas, the way we categorized reactions and pattern of collaboration was not based on literature, and the patterns were not given special attention either. Now, in our new schema, they would be important parts of the collaboration assessment.

7.5.2. Initial Decisions

The new coding schema's categories had English names and descriptions. Our previous coding schema versions were all written in Norwegian, but translated as attachments for this thesis. The English categories would be easier to relate directly to the literature, since the literature was English. Also, the prospect of involving non-Norwegians into the coding work was mentioned, so it was best to make the categories English sooner rather than later. Since this schema revision was a large remake, it was a good idea to change the language now, since all categories were revised during this process anyway. We would risk a decrease in clarity to the definitions if they were translated separately.

Since the new categories required much more effort to apply to the audio files, it was decided that we would code only task 4 of the experiment, the largest non-time-sink task. We also decided to omit the discussions between the programmers after each task from the coding. In these, the programmers decided how good they thought their collaboration had been during the task, and then filled out the forms about the collaboration and how much time they had used on the task. These sections of the dialog were not about solving the programming tasks, and were considered as unimportant for us to focus on.

7.5.3. Tests and Implications

Our new category scheme, the fifth in line, was never used in more than very short tests, but we discussed it a lot, and it was the first of three quite similar schemas, of which the third would be our final version.

Initial tests our or new, fifth, schema led to a couple of additional decisions for our future coding process. First of all, all parts of the protocol for task 4 were supposed to be part of one clip – nothing more, nothing less. Previously, there had been no formal decision on this. Some of us placed the time markers in Transana before each clip only, while others placed markers both before and after clips, sometimes leaving tiny clipless time areas between clips. We decided that we would make time markers in front of clips only, in order to avoid clipless time frames. It was better to make incredibly short clips coded as *silence*, than to have a "hole" in the coding.

However, a little later, we decided that periods of silence had to be long enough as to break the flow of the dialog to count as silence. Pausing for a couple of seconds for taking a breath or considering what word to use would therefore not be enough for us to make a "silence" clip. We distinguished this kind of silent situations from "real silence", where both programmers were silent because there was nothing to say or one of the individuals in the pair were not present at the time because they took a break, etc.

7.6. More Development of New Categories

Only a week later, further development of the categories and their structuring led to the declaration of a new coding schema version, the sixth. While not radically different from schema number five, number six improved on a couple of potential problem areas.

First, the number of subcategories to be applied to the protocol, (for the clips containing an interaction sequence,) was now increased from five to six. Though this might sound like a change making it harder to do the coding, it actually made the coding easier. The increased number of subcategories was the result of the splitting of the *result* subcategory group from schema 7.5. The first of the two groups formed by the split was named *end*. It described how the interaction sequence ended. Was there a natural transition to the next topic or another action, for example programming, or was there a disruption or perhaps a sudden change of topic? The second of the two subcategory groups formed by the split, kept the name *result*, and kept its definition as well (see section 7.5.1). The previous schema version would make it impossible to distinguish a clip that ended with a disruption that did not solve the problem, from a clip that ended naturally but did not solve the problem, if the coder, using the fifth schema, chose to focus on the problem solving, and coded both sections as *unresolved*.

The category *Stonewalling* was added to the interaction pattern category group, and was meant to be used in situations where *unresponsive* was not completely suitable, for instance if there was an answer to the initial comment, but the answer disregarded the contents of what was said. Stonewalling is arguably the only category we had left that described a behavior that was quite "impolite" (see section 7.2.4.). The category was inspired by the coding schema of Chan [Chan 01], and was thought to prove to be very interesting regarding personality. Many people would probably never perform stonewalling-like actions due to their personality or other background factors.

This schema version can be seen below, in Figure 8.

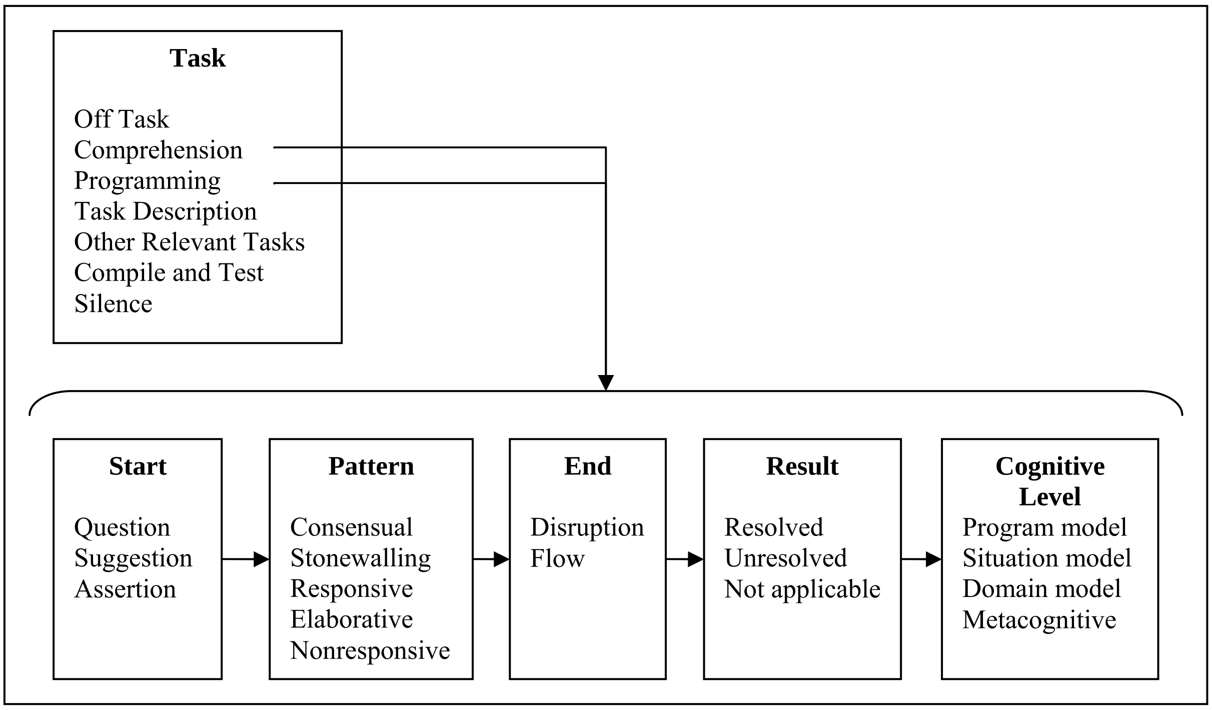

Figure 8: The further developed new focus category schema

7.6.1. Usage

When this schema version was declared as complete, we all coded the same fifteen minutes of a certain audio file using the new schema. When using the earlier versions of a schema, it would rarely be needed to rewind the playback of the audio files in order to code a clip correctly. For our new schema, however, it was necessary to do so, often many times for a single clip, in order to grasp and successfully code all the six areas of focus we now had, and to get the start and end time marker placed as accurately as possible.

During the coding comparison meeting afterwards, it became apparent that the categories themselves were sufficiently defined, since clips of similar areas were coded very similarly also. The problem remained the positioning of the time codes making the clips, which still deviated quite a bit when comparing our individual coding of the file.

Also, for some categories we had slightly different opinions on the definitions, since a firm list of category definitions was not made for the new categories at this point. After discussions and consulting the literature the categories in question were based on, agreements were made for all the major problem categories.

7.7. The Development of the Final Schema

After tests and experiences with the 7.6 schema, we realized that we needed a category that could describe clips where it was impossible to hear what was said, since it was unfortunate to be forced to use some other categories when one could not know what happened. The new category *unintelligible* was made. It was meant to be used when after many tries, the statement or discussion area in a clip was still not clear enough to the coder in order be able to categorize it correctly. There could be several reasons for unintelligible situations, including murmuring by the programmers, background noise during the clip or simply low quality of the recording for that particular pair.

A new category in the *end* subcategory group, named *divergence*, was added. It was meant to be used in situation where the patterns *stonewalling* or *nonresponsive* were used, to indicate that a new problem was introduced during the clip that was not intruded by the start statement of the clip. It was realized a little later that this category fitted better among the *interaction pattern*s rather than among *end*s. We found it better to be able to decide whether the initial problem was solved or not regardless of the discussion pattern, and rather make an extra clip right afterwards that could indicate that there was a divergence. Divergence was later renamed to *cross-purpose* in the dialog, but its definition remained the same.

7.7.1. Reformatting

A little later, the coding schema figure was thoroughly reformatted by the supervisor, in order to tie the schema even more in towards the literature and the concepts used in the articles. Minor alterations were also done to the categories. Also, all categories were given a thorough definition.

The resulting schema and its definitions would turn out to be the final version of our coding schema. The schema itself it described in more detail in Chapter 8.

7.7.2. Testing and Early Experiences

We decided that all people involved in the project would code parts of a certain audio file. The correspondence between how we coded the file, when we compared our results was very high. The categories used were usually the same or at least very similar between all of us. Even the time of the clips’ stars and ends (which was previously thought to be very hard to agree on when done individually) matched satisfactorily at this point.

8. Our final Coding Schema

8.1. Description

The final coding schema is complex and is quite time-consuming in use, but it has many foci, which will lead to much analysis material. The schema was refined many times in order to make it suitable to use for coding the collaboration and activities most frequently used by the programmers in the pair programming experiment (Section 1.3). The schema is also flexible enough to be used to describe unexpected situations.

The coding schema is thought to be both mutually exclusive (because of the categories' non-colliding definitions), and exhaustive (see Section 5.2.3) for each of the six category groups. Exhaustiveness was improved when the categories *Not applicable* and *Unintelligible* were introduced.

Since the clips are assigned to the six category groups one by one, and the groups are independent of each other, coders are not forced to choose a certain category based on his or her previous category choices for the clip. To exemplify; if the coder chooses *Comprehension* as *Task Focus*, it neither forces nor restricts him from choosing *System Model* as C*ognitive Level*. It does, however, restrict him from choosing *Programming Aloud* as the *Task Focus*, since he has already chosen a *Task Focus* category. . Also, there is one exception regarding the independence between the categories: if a Task Focus other than Programming or Comprehension is chosen, it means that the clip should not be assigned to categories from the remaining five category groups.

Clip lengths when using this schema are typically medium in length compared to the clips commonly found in the available literature. The clips will not consist of just one word, (like the schema in Section 6.3), but they would not last for several minutes either, like the clips often did when we tested early versions of our schema. Clips coded as *Programming Aloud* will sometimes be of great length compared to the length of the discussion-based clips. Since these long clips are monotonous in content, it will not lead to any more interesting results if one divides them up into several similar-coded clips in a row.

A possible weakness of the schema is that it demands subjective decisions by the coders in certain situations. *Result* is the subcategory group for which this problem could be most apparent; it is sometimes a little unclear whether or not the course of action is changed because of the idea just presented or not. However, because of the coding schema will be thoroughly reliability checked and the file coding calibrated, these subjective characteristics of the schema are not considered as a significant problem. The calibration and reliability calculation are described further in Chapter 10.

As described earlier, for each clip, a *Task Focus* will be chosen. If the T*ask Focus* is *Comprehension* or *Programming*, the clip is an *Interaction Sequence*, and the rest of the six subcategories (*Begin Characteristics, Interaction Pattern, End Characteristics, Result and Cognitive Level*), will be selected.

The figure below (Figure 9) lists all categories and the subcategory groups in which they belong. The lines from the *Task Focus* box indicated which two *Task Foci* that will lead to the complete *Interaction Pattern* coding.

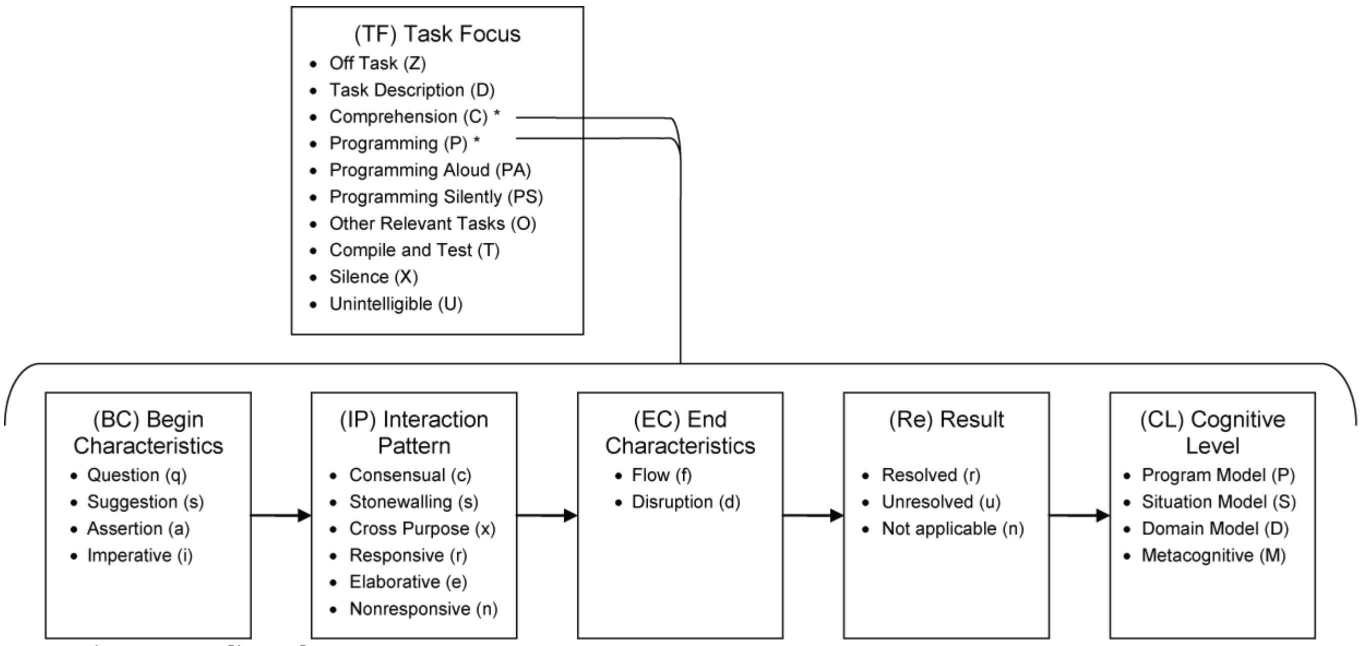

Figure 9: Coding schema

8.2. Category Definitions

8.2.1. Task Focus

The *Task Foci* describe *what* is being done or discussed by the programmers.

Z – Off-task – For discussions about things completely outside the experiment, for example discussions about current events.

D – Task Description – When the discussion is about the actual task description itself. Reading the text out loud is an example. When it is likely that both programmers are reading the text, but they are not discussing it, X (silence) is used, not D. Any specifics about what should be done, other than very high-level structure decisions based directly on what is read in the description, should be C or P.

C – Comprehension – When the programmers are discussing existing code and what elements of it mean or do. Not to be used on discussion about what should be done next, where P is the correct choice.

P – Programming – Used in instances where the discussion is about future directions for solving the task. Must contain suggestions, assertions or discussions about what is to be done. When discussing previously produced code, and how these things work, C is the correct choice.

PA – Programming aloud – For situations where actual programming (keyboard use) is audible, or where it is completely obvious that programming is taking place. To qualify as PA (separating it from PS) at least one of the programmers must talk sporadically or continuously throughout the clip, for example by saying out loud what they are writing, or commenting and pointing out objects on the screen. Discussions with more substance than this should be described by using P instead.

PS – Programming silently. For situations where actual programming (keyboard use) is audible, or where it is completely obvious that programming is taking place, but there is no talking going on.

O – Other relevant tasks – When the discussion is about solving the experiment, but not directly about the programming tasks.

T – Compile and test – This category is used when it is clear that the programmers are compiling or testing the program and walking through the menu selections of the running program or tracking the lines that the compiler state as erroneous.

X – Silence – To be used when there is no dialog at all. Sighs and sounds which have no obvious collaborative purpose can also be categorized as silence.

U – Unintelligible – Used when the dialog is unclear enough or the sound quality is so low that it would require guesses to put categories on this section of the file.

8.2.2. Begin Characteristics

The *Begin Characteristics* describe how the interaction sequence was initiated.

q – Question – Used when one is requesting information without adding any new information. The question can for example be about how a certain method works. "What do you think about this", and similar requests for feedback on ideas presented are not "question", but "suggestion".

s – Suggestion – Presentation of an idea in the form of a question in words used and tonality. "Should we place a new method here?" is an example.

a – Assertion – Suggestions made more directly, presented as a fact or a course of action. "Let's make a new method here" and "Ok, so then we need a new method" are examples.

i – Imperative – Suggestions for course of action more similar to orders than the assertions and suggestions, but not necessarily aggressive.

8.2.3. Interaction Pattern

The *Interaction Patterns* describe how the dialog is after the initiation (the *Begin Characteristic*). How does the non-initiator respond? And does the initiator respond to the response?

c – Consensual – The responding part does not contribute any new information, but acknowledges the statement just presented.

s – Stonewalling – The responding part ignores or quickly refutes the previous statement.

x – Cross purpose – If the initiating person continues to talk about his line of action/thought/dialog, and the responder talks about his unrelated ideas, and both are talking about their different things at the same time, the pattern is a *Cross Purpose*.

r – Responsive – The responding part answers by adding some new information. It could evolve into an elaborative pattern, if the initiator responds with new information again.

e – Elaborative – The responder and the initiator gradually build up the understanding of a topic by both contributing with new information.

n – Nonresponsive – A statement followed by complete silence from the other part. A mumble or acknowledging grunt is enough to make it consensual, rather than nonresponsive.

8.2.4. End Characteristics

The *End Characteristics* describe how a clip ends.

f – Flow – When the discussion ends "naturally", and without interruptions or sudden topic changes.

d – Disruption – If someone changes the topic before the dialog naturally evolves by its own momentum.

8.2.5. Result

The *Result* category describes whether or not the discussion leads to a future course of action or consensus or not.

r – Resolved – Used when the initiating statement is sufficiently discussed and ended, or when a course of future action is set.

u – Unresolved – The opposite case of *resolved*; when issues are still hanging in the air, agreement cannot be made, or misunderstandings persist.

n - Not applicable – When none of the above are suitable, for example in situations with cross-purpose dialog, and both are satisfied with their own ideas, but none of them have heard the other person's comments.

8.2.6. Cognitive Level

The *Cognitive Level* describes which level of abstraction the dialog is on. Since the experiment was performed on a problem that most (perhaps all) would be familiar with (a coffee/drinks dispending machine), and since the programming language in use was the object oriented Java, it would be likely that the discussion would range from the very programming specific, and all the way "up" to the conceptual level of the coffee machine.

P – Program Model – Discussions about the code itself. Formatting issues, syntax, library uses and basic operations are examples of discussions about the program model.

S – Situation Model – Discussions about the interconnections and big picture of the system. Talk about classes and the flow of methods and calls are examples of this.

D – Domain Model – Discussions that link the system to the real world or talk about the physical object in which the system is supposed to function.

M – Metacognitive – Discussions about understandings of the task and how it is solved and other situations that are neither of the above.

9. Coding the Audio Files

9.1. Sample Selection

The first files we coded were selected randomly. When we a little later decided that we would not code all the audio files we had available due to time constraints, a selection of some of them had to be done. The pairs with high or low expertise levels (not medium) were prioritized. Also, the pairs consisting of Brits were avoided, due to language differences, and the fact that the coders were all Norwegian of nationality, and had Norwegian as their first language. Since we initially chose some audio files by random, two British pairs and six intermediately skilled pairs were included in our selection. In total, we coded 47 audio files, which mean 47 pairs. We later found out that for three of the codings, task 5 had been coded instead of task 4. We omitted these codings, since the collaboration in the longer time sink task, task 5, was probably not directly comparable to that in task 4, so our final sample size became 44.

9.2. Hired Help

The process of coding the audio files was a large and repetitive job. Efforts had been made to make the coding schema as objective as possible, meaning that anyone (assuming they had the proper training and insight into the definitions of the codes) could do the job. For this reason, a number of Simula people were hired by to help us in this.

9.2.1 The Task 4 Finder

One person worked exclusively on making a time marker for the audio files that marked the point where the programmers in each file started their work on task 4 (which was the task we coded). Finding these starting points was reported as taking 10-20 minutes for each file. The start of task 4 was identified for many, but not all, files. When we, the coders, had to find the starting points ourselves, it sometime took us much longer than 20 minutes. It was an especially time-consuming job on files that were generally low on audio quality and for the files in which the programmers followed an unconventional work pattern, and took unexpected or long breaks.

For one audio file, I had to abandon trying to find task four, since hours were spent looking for it, but no task four could be found. Presumably, the programmers unknowingly skipped the task due to misunderstandings or erroneous use of the forms they were meant to fill in. The whole time between the end of task three and the apparent start of task five, the audio file was filled with indications of the programmers taking a break, and there was no relevant programming discussion anywhere. It is possible that the programmers in the file could have solved the task during the long section I thought was a break, but considering that task 4 was the hardest of the non-time-sink tasks, it is unlikely that they felt they had no need to discuss it at all, especially since they discussed the previous tasks (which were much easier) as thoroughly as most other pairs did.

We, the coders, checked a random selection of the time codes while they were in the making to confirm that they were actually indicating task four. We found that the marking of the task was spot on for all files we checked. The work of the start 4 finder was valuable help that saved us a lot of time.

9.2.2. The Audio File Coders

Two additional people worked as coders, and followed most parts of the procedure described in Section 9.3 below. Reports from the two hired coders indicated that it took them roughly between two and three hours to code the shorter files and those with the best quality, while the longest files, and those with worst quality, took the whole workday to code.

9.3. The Coding Process

After spending one day programming the tasks in the experiment ourselves in order to get more accustomed to the terms and concepts that were relevant for the programming tasks, and to experience ourselves what the challenges of the tasks were, the first of many audio files were coded using the final coding schema.

The coders made clips where appropriate and assigned these clips to the relevant categories by writing the categories as text in Transana. Mostly, we needed to rewind and listen to the clip several times, since in many situations it was hard to grasp the complete set of the six category group foci when only one person was doing the job. Also, placing the start markers of the clips often required several rewinds in order to get right.

An example of one coded minute is shown below. Most of the text is automatically generated by Transana when making a time marker, which we did with the convenience of the CTRL-T hotkey combination on the keyboard. The ¤ and the number following it is the desisecond of the file this time marker is placed on. Behind the desisecond time, in parentheses, is the actual hour:minute:second:desisecond time shown. To show both these time codes is an optional feature of visual significance only, and the latter time code is not necessary for Transana to function, but the time listed in the latter format is surely more understandable for most humans).

After the clips were made, what remained to be done was to enter appropriate categories for the clip. In the example below, the first clip (starting at 1:07:54.7) is an interaction sequence where the topic of discussion is programming related (P). The clip is initiated by an assertion (a), the response by the other person is merely consensual (c), the discussion flows to a natural end (f), and the issue is resolved (r). The discussion is primarily on a low program-level abstraction (P), and the big picture of the system is not discussed in the clip.

Further down, we see clips coded as *Programming Aloud* (PA) and *Off-task* (Z). Since only the task foci *Programming* and *Comprehension* was regarded as relevant for analysis on interaction sequence level, these two clips are categorized only by the first category, the task focus.

```
¤<4074660>(1:07:54.7)PacfrP
¤<4083246>(1:08:03.2)PqrdrP
¤<4092154>(1:08:12.2)PqrfrP
¤<4095732>(1:08:15.7)PsrfrP
¤<4106944>(1:08:26.9)PA
¤<4119102>(1:08:39.1)Z
¤<4135375>(1:08:55.4)PanfrP
```

Coding Example

A couple of extra procedures, both described below, were carried out for every file that was coded in order to increase the quality of the coding and ensure that our coding schema was of sufficient quality.

9.4. Parsing

Originally, we planned to use Transana's clip functionality in order to define clips and assign these to categories, since statistics about the categories could then easily be generated by Transana. However, it turned out that to actually define Transana clips and assigning them to categories in Transana was so time-consuming now that each clip was meant to be assigned to six subcategories, that the idea was abandoned. Even though Transana was still a valuable tool for making time codes and manipulating the playback of the audio files, we wrote the categories simply as text in the transcription area, rather than using the categorization system in Transana.

Because of this procedure, there was need of a way to convert the text files that we made in Transana into aggregate numbers that could be imported into the statistical software we would use in the analyses later. Initially, our idea was to import the text files into MS Excel and make some scripts or use the database functions of Excel (or MS Access) in order to convert the text into category usage numbers. However the Excel idea was soon abandoned, since it would not have been easy to do it that way. We did not think through that the time code in front of the categories was the time marker where the clip started, **not** the duration of this clip in question. One had to calculate the duration for each clip by comparing the clip marker in front of the clip and the clip marker on the next line (in front of the next clip). It seemed that using Excel or Access for this would be cumbersome.

To solve the problem, I programmed a parser in Java. It was a quite straightforward parsing program, which read the Transana output files. It required no manual input, and did the work as fast as one would expect; in a matter of seconds in total for the whole lot of files. The program found the time codes in front of each clip and calculated, based on this time code and the one for the next clip, the length of the clip. The program then organized the categories and made statistics about their use in each file.

The output of the program was double. One output was made for each input Transana file. These outputs were used for debugging and for controlling that the individual files were read correctly. The second and most important output was a large table consisting of all categories and all audio files, where the use of categories was columns and each input file was a row.

The program calculated, after reading through all the files, which of the possible *long categories* that were used, and included only these in the outputs. Long categories are the composite of all the six subcategories that describe an interaction pattern (See Chapter 8). Initially, an idea of making outputs for all possible combinations of categories that formed long categories was planned. However, it was soon realized that this would mean a table with more than 10 000 columns, since each of the possible 5 760 long categories would have at the very least two columns each, and preferably five. Most of these long category combinations were never used, so to have an incredibly large file consisting of mostly empty cells was avoided by making the parser slightly more complex instead.

After some feedback and discussion back and forth between my supervisor and me regarding the output format and the compatibility with the potential statistical software to be used, the parser was completed. The parser was not meant to be a robust and highly elegant piece of software, but to solve a quite simple task that would be tedious and time-consuming to do manually, as quickly as possible. Therefore, its Java code is not especially easy to grasp for most people who was not its developer (i.e. anyone but me). Also, the program is not very tolerant to what it sees as “errors” in the input files (the output files from Transana), and a single space character in the wrong place could mean the difference between a successful parsing run and not. However, the parser program had quite informative error messages and alternative outputs, so in cases where the parsing did not work correctly, it was easy to alter the input files in order to make them correctly formatted. About a third of all the .rtf files turned out to have one or more such small errors, so before I could do a successful run of the program and generate proper outputs, I had to correct them. But to correct them was necessary only once, and it did not take more than a few hours to do so.

10. Reliability of the Coding Schema

Because of the inevitably subjective elements of the coding process, there was need for some sort of control that the same categories would be used in the same situations by two different coders. Even after all the revisions of the coding schema and the attempts to make it as un-subjective as possible, the schema was not maximally objective. We had to keep its objectivity within limitations so that it did not require a week of work to code one file, as would have been the case if we had used schema more similar to that described in Section 6.3 above.

Because of the low quality of some of the audio files and differences in how the coders interpreted situations every now and then, it was unrealistic to hope for a 100% match when comparing two coders coding work on the same file. For example, it is not always easy to hear who is who of the two programmers if they have similar voices. When you don’t know who is saying what, it is hard to know if a person continues a line of thought or whether the other person answers him in some situations. However an agreement score of at least 70% seemed like a realistic and adequate goal.

10.1. Calibration

Bakeman and Gottman [Bakeman 97] mention calibration as a good idea when several coders are working with the same coding schema on the same material. We used the following calibration method: For each clip, two people coded the same pre-selected area with a duration of five minutes. This was typically five minutes early in the task (Task 4), either the very first five minutes of the task, or the five minutes starting at the five minute point into the task, so that the area was from minute 5 to minute 10 of the task. After coding, the two coders would meet and discuss the coding on these five minutes while listening through it and comparing their individual coding of it. Through these discussions, they would modify their coding into one coding that they agreed completely on. Later the *owner* of the file (the person who was responsible for coding the complete task) would look through these five minutes before starting to code.

The purpose of the calibration procedure was to gain two benefits. The first and most important benefit was that it would make the owner of the file better understand the dynamics of the pair in the file he was supposed to code later. The pairs had slightly different typical patterns of interaction and informal rules for when to speak and whether the driver or the navigator was the one supposed to “lead” the development. When five minutes of the file was discussed thoroughly to make sure these minutes were coded as accurately as possible, it was thought that it would be easier to code the rest of the file. The idea was that the file owner would get “the gist” of how the pair behaved by listening through these five minutes and looking at the hopefully perfect coding of them.

The second benefit was that these five minutes of calibration meant five minutes of “free” code reliability check material (see Chapter 10), since two people would coding the same section of the code during the calibration. For the part of the reliability checks that were not based on the calibration, a person had to do coding work that was of no other value than to provide comparison data for reliability testing of the coding schema. However, when two programmers coded the same area during the calibration phase, the data was used for the actual calibration as well. This double use of the data from the calibration made the procedure clearly worth the effort.

The two hired coders did the calibration with fully individual coding on all their clips. The other master student and I did the same with the first third of the files we coded. One could see this as the ideal calibration method, since the first coder could not influence the other one in any way before the meeting where they discussed and compared their coding. However, the method proved to be less than perfect, since the discussions in the calibration meetings turned out to be about the exact placing of the time codes rather than about the choice of categories.

For the last two thirds of the files, the other master student and I used a slightly altered calibration procedure. In our new procedure the owner of the file first coded the five minutes to be calibrated, then removed the categories from the text, but let the time codes remain, and then sent this “clip skeleton” to the other person. The other person, based on these time codes, added appropriate categories to the already existing clips.

For record keeping and code reliability check purposes, all pre-calibration files were kept, and none of these were edited without keeping a copy of the original.

10.2. Reliability Checking

After the calibration, the file owner did the full coding of task 4. After coding the task all the way through, the file owner sent the text file containing the coding of the file to the community of coders, (the two hired helps and the two master students.) Then, one of the two master students would check five randomly selected minutes of the file. (The randomization was done by a simple program described below.) If one of the master students was the file owner, the coding would be checked by the other master student; the one who was not the file owner.

This reliability check would not be like the one in the calibration; where two independent codings were made and compared. In the reliability checking, the job of the checker would be simply to control whether or not he felt that the code applied to the five minute section to be checked was correct. For clips where the checker disagreed with the file owner, the checker would write his own coding suggestion behind the original code. As with the calibration, in order to have a complete record of files and versions, no files were altered without keeping a copy of the original.

The randomizer that selected what part of the coding that would be reliability checked was a simple program I made, where the inputs were the start and the end times of task 4. The randomizer then calculated, using a standard library random method and the input timestamps, when the check was to begin. The program did take into account that the first ten minutes were not to be checked (since parts of these minutes were already checked during the calibration phase). The program also avoided including the last five minutes in the check area, since these minutes most often consisted of long clips of *Programming Aloud* or *Compile and Test*.

10.3. Reliability Calculation Approaches

Several approaches was suggested and tried in the process of making a good system for the calculation of the reliability score. They are described in the three sections below.

10.3.1. Initial Approach

The first reliability calculation method we suggested was the simplest one. One person would go through another person's coding of a file and write down whether he agreed completely or did not agree completely on the coding of each clip. The number of disagreements would be divided by the number of clips checked. The checker would also check whether or not he or she agreed with time marker placement of the first and last clip of the task.

While the method described here would be quick and simple to perform, it would lead to results that were not as "fair" as they could have been. For example, if following this method, a clip coded as X (silence), where the checker suggested PS (Programming Silently), two categories both being known for their lack of conversation, the agreement would be given a score of 0. Similarly, a clip coded as X where the checker suggested using a long interaction sequence category, for example PaefrP would also receive the agreement score of 0. It was not ideal that those two situations, where the first one was probably nothing more than a misinterpretation of sounds, while the other one was a clear error in the coding, would be regarded as equally bad.

10.3.2. Stricter and More Individualistic Version

Later, it was suggested that we should do a completely individual coding of the area to be reliability checked, similar to what we did during parts of the calibration. This way of doing it would be the one with less risk involved of getting sloppy reliability checks, since there would be no information about how the other person had done it. However, as during the non-skeleton based calibration, this double coding turned out to differ quite a bit on clip placement. Therefore, a reliability check would require an additional cooperative walkthrough of the checked area after coding and checking where both people involved (the coder and the checker) discussed it and agreed, much like in the calibration phase.

It was soon realized that this procedure would not be used by us since it would be negatively disproportionate regarding how much work is would require compared to the value of what it lead to. For a complete check of the whole code, where the objective would be to make sure the coding of the file was as correct as possible, this procedure would have been great, but for our purposes; checking whether the code schema was sufficiently objective; it would be excessive.

10.3.3. Subcategory Based Approach

Bakeman and Gottman [Bakeman 97] state that one cannot successfully calculate agreement simply by dividing the number of agreements on the total number of units. One must define more specifically what an agreement is, and what a unit of agreement should be.

Based on this and, as the next possible approach, we came up with something completely different. This time, we would compare each subcategory used by the coder with the suggestions of the checker. A fraction score would be given based on how many of the subcategories that matched when comparing the coding to the checker's suggestions. A clip coded as **Par**frP would lead to an agreement score of 3/6 (50%) if the checker suggested **Par**duS. The **Par**frP coded clip would give a score of 0 if the checker had suggested CscduS, as none of the six subcategories matches. After comparing each clip and calculating for each of them a fraction score, an arithmetic mean would be made for the scores of the clips checked, and this number would be used as the measure of agreement for the coding of this pair.

For clips where the coder used a long category and the checker suggested a *short category* (i.e. a task focus other than P or C, where the task focus alone would be the complete coding of the clip), the calculation would be less straightforward. Special rules were made for these situations. These rules are listed in Appendix X2.

By using these rules, two files were compared as part of the reliability check. The first of the two files consisted of the five minutes the file owner coded before the calibration, plus the five minutes the coder had coded further into the file, that was selected by the randomizer as the reliability check area. The second file involved in the comparison was the selection of the same clips that was included in file number 1, but where the coding of the clips was as they were after the calibration, and where the checkers's codes had replaced the original coding for the five randomly selected minutes.

10.4. Reliability Calculation Results

The reliability/agreement scores for each coded file were aggregated and a number of measures for reliability were made based on them. Our primary measure of reliability is given by the reliability check rules described above and listed in Appendix X2. By using its rules, a score of 94.24% was found as the mean for all the pairs. The standard deviation was small: 3.92 points, so the great majority of files coded had a reliability score in the nineties.

The mean amount of clips in the ten minute areas included in the reliability check was 30.36. The mean amount of clips per pair in which the two coders (the coder and the checker) were not in agreement regarding the start or end time marker of the clips was 2. In percentage, this results in an agreement score of 93.72% when measuring clip time agreement. The mean amount of clips where there was some dissimilarity between the two codings was 4.86. Therefore, the percentage of clips with a *total coding agreement* (where the categories used matched completely) was 83.15%.

The lowest score on agreement between the coders was found when I calculated the total coding agreement plus the total agreement regarding the starting/ending points of clips, and compared this number to the total number of clips. This led to a score of 76.88, which was also reasonably high. However, that number was not highly dependable as a reliability measure, since for some of the clips for which there was a disagreement on the timing of the clips, there was also a disagreement regarding category use. Therefore, the 76.88 score might have been a little lower than the reality.

When using the total coding agreement and the clip time agreement scores mentioned above together, that is, when calculating 93.72% of 83.15%, the result will be 77.93%, possibly a better representation of the agreement than the 76.88 score above.

When I used the agreement score calculated percentage (94.24%) instead of the total coding agreement score (83.15%), the score like that above (based on combining the percentages for clip and category disagreement) was 88.33%.

All of these numbers are summed up in the following table, Table 5.

#	Reliability Calculation	Score
1	Reliability Check Rules	94.24
2	Clip time Agreement	93.72
3	Category Total Agreement	83.15
4	Clip + Category Total Agreement	76.88
5	2 & 3	77.93
6	1 & 2	88.33

Table 5: Code schema reliability results

Personality and Pair Programming

We all know the visual stereotypes of a programmer. He is a man wearing glasses and perhaps a geeky t-shirt with the "There are 10 types of people in the world – Those who know binary and those who don't" joke, or a motif from a game or a cult film on them. He drinks coffee or Coca-Cola all day, and has a dislike of sports and other physical activities. He is also a quite obvious and outspoken fan of certain movies, and it is not unlikely that he can (and often will) recite long sections from Star Trek, Star Wars or the Monty Python movies. Although most of these are things that one can change quite easily, some people like to belong to this stereotype, and would rather increase than decrease their similarity to it.

Another side of the computer geek stereotype cannot be easily altered by programmers themselves. These are the more internal factors of a person, including the personality. As stated in [Hannay 09a], programmers are indeed significantly more introvert, neurotic and open to experience than reference groups.

The goal for this thesis is not to identify the personality profiles that are most common in programmers; this has been investigated quite thoroughly already. In addition to the results found in the pair programming experiment, [Hannay 09a] mentions four other investigations into this. One of them is an MBTI-based paper which finds that a majority of "systems analysts" are introvert, and a great majority of them are of the TJ (thinking and judging) type, which the authors of the article [Smith 89] describe by making a citation:

> These individuals are described by McCaulley (1978) as tough-minded people with a greater aptitude for technical as opposed to interpersonal skills. They are also logical, analytical, responsible, dependable, systematic, decisive and organised. They plan their lives carefully but are somewhat inflexible.

The goal of the thesis, however, is to identify which personality traits that affect collaboration, and in what way. It might be the case that the "stereotypical" programmer is the one also most suited for collaboration in pair programming, or he might, due to his low extraversion, be the worst. Would two similar people personality-wise collaborate differently than two more different people? These questions and more will be investigated and answered in the chapters to come.

11. Analysis Considerations

11.1. What Variables to Base Analyses On?

With 285 data columns for each and every pair gathered during the pair programming experiment, and several hundred columns of category data generated by the coding parser program described in Section 9.4, the decision on which of these variables to include in the analyses was not immediately obvious. The selection process is listed in the sections below.

11.1.1. Personality

For personality, the choice was easy enough. During the pair programming experiment, statistics were recorded about each person's personality scores on the Big Five factors. Since we had decided earlier that the pairs were the focus point; not the individuals, the columns with the combined results for the pairs were used. For each of the "big five" personality traits, each pair's mean score was listed in a column in the data table from the experiment. These columns were selected, and would show if a pair as an entity scored high or low on a given trait.

Each of the five traits also had a column where the difference in score between the two pair members on that given trait was noted. This column, titled "trait"_StdDev was also used. It would be needed when investigating whether similarities or differences in personality traits internally in a pair had any influence on what collaboration categories we used to describe the collaboration of this pair.

The two personality columns we chose would inevitably have a slight correlation. Mean scores close to zero or a hundred would mathematically mean a low StdDev, since the trait scores for the big five traits are percentage based. If the mean is 99, the StdDev could maximally be 1 (since the maximal score is 100, and (100+98)/2 = 99). It is likewise for the other end of the scale: A mean of 10, would mean a maximum StdDev of 10 (since (0+20)/2 = 10). However, for medium mean scores, which will be quite common, there will be no logical correlation like this between mean and StdDev.

11.1.2. Collaboration

The parser program generated huge amounts of data, since it was programmed to write out five columns for every category that has been used during the audio file coding. Since each long category describing an interaction sequence consisted of six subcategories, and there were quite a few subcategories to choose from for each of these six, the amount of subcategory combinations that were used was large. Luckily, the number was nowhere near the amount it might have been if every single possible combination of subcategories into long categories were used.

A little later, it was realized that analysis based on the combined interaction sequence categories would not be ideal, since they were so numerous and most of them were used very rarely; sometimes just once in one of all the coded audio files. To look at the subcategories individually would be more manageable, and would give us just as good material to make statements and hypotheses on collaboration with.

After that decision, there was still the issue of which of the five fields for each subcategory that would be the focus. For example, for the *Task Focus* category "Programming" (TF-P), the five columns were named TF__P_no_of_times_used, TF__P_time_used, TF__P_p_of_clips, TF__P_p_of_time and TF__P_average_time. They are described and discussed below with regards of usefulness in the analyses.

11.1.2.1. no_of_times_used, time_used

No_of_times_used, which indicates how many times the subcategory was used during task 4 of the audio file, was not suitable for our analyses. The numbers in this column would depend heavily on how long it took the pair to complete the task. A long task 4 would typically consist of more clips than if the pair had solved the task faster. When the clips were more numerous, some categories were bound to be used more often than they would otherwise have been. No_of_times_used would therefore not provide sound data to base analysis on in itself. The same applied to the similar column time_used.

11.1.2.2. p_of_clips

p_of_clips, short for "percentage of clips", indicates how often the category was used when compared to the other categories. At first, it seemed to be a realistic candidate for analysis. However, the column in its pure form was ultimately not chosen, because of its comparatively high vulnerability to variance in how the categories were used. Inevitably, each audio file coder would have a slightly different opinion on when it was appropriate or not to make a Z (silence) clip. Even though we decided how long a silence would have to be in order to be made into a clip, factors such as "how silent" it was, and whether or not the silence was slightly longer or slightly shorter than this limit, would lead to a differences in use of the category Z, and thus the number of clips two coders would make for the same file. The percentage of clips column would be affected quite noticeably by this difference.

11.1.2.3. average_time

Average_time would not be affected by short clips the same way as p_of_clips, but average_time would also not describe the collaboration in the ways we wanted to investigate. Even though it could be interesting to see how long people stayed silent, or how long they elaborated on each other each time it happened, it would be unsuitable to make analyses based on this column. If a pair was silent very often, for thirty seconds every time, the pair would be identified as just as silent as a pair who was silent only once, for thirty seconds. To classify these two pairs as similarly silent would not be ideal.

11.1.2.4. p_of_time_used

p_of_time_used contains the percentage of time the category was used relative to the total time of the solving of task 4. For this column, the problem with p_of_clips; that coders could make different amounts of very short clips, would not be important. If a coder made ten extra silence clips, but each had a duration of only one second, it would not change the percentage of time used for the other categories much, since a single normal clip is mostly longer than ten seconds in duration.

11.1.2.5. New Ideas

After a brief analysis run using p_of_time_used, it was realized that p_of_time_used also was not the ideal column to use as it was, in its pure form, since the column had a couple of imperfections. First of all, the numbers in the column was generated simply by dividing the time use for the category on the total time used for solving task 4. While this was the most obvious way to make a column of the percentage of time, it was not excellent since it would include the clips labeled U for *Unintelligible*. For audio files of especially low quality, the number of U clips was quite high, and therefore the percentage of time used on all the other categories would be lowered because of the frequent U use.

Also, the clips consisting of interaction patterns/sequences would best be compared only to the other interaction pattern clips. When using p_of_time a pair which programs loudly the whole time, except for in ten clips, all of which begins with a suggestion, would receive a very small percentage use for suggestion. It would "drown" in the heavy use of programming aloud, even though it was used much more frequently than the other B*egin Characteristics*. Since the *Task Focus* categories that did not imply an interaction pattern was quite frequently used, it was more accurate to compare interaction pattern components to other interaction pattern components only. The triggers for interaction patterns, the task foci of *Comprehension* or *Programming*, would still be compared to the audio file as a whole, so one could easily know how frequent interactions patterns were used.

11.1.2.5. New Columns

The parser program, described in Section 9.4 was modified to include the two new categories described above, titled p_of_nonU_time (percentage of time this category was used compared to the total time used on the task when the category used is not U) and p_of_ip_time (percentage of time this category was used compared to total time for all clips coded as interaction patterns). For the task foci, the p_of_nonU_time column was used, for every other subcategory type; the p_of_ip_time was the appropriate column.

A new run of analysis was performed with these new columns used instead of p_of_time. Since it was at this time that we discovered that some of the audio files has been erroneously coded (task five instead of task four was coded), this new round of analyses was not an addition, but a replacement of the previous runs, since the data sample was different.

11.1.2.6 The Return of p_of_clips

A little later it was realized that p_of_clips could be of use after all if one used the same line of thought as in p_of_ip_time, and counted only the interaction pattern clips. This way, short silence or silent programming clips, which are not interaction patterns, would not distort the results. A new run of the interaction pattern analyses was done with this remade version of p_of_clips. This time, the results were based on the same material as last time. Therefore, it was appropriate to view the new results as additional material, not overwriting the findings from the analysis runs based on the factors described above.

11.1.3. Performance

Regarding how the pairs performed when solving the tasks of the pair programming experiment, the different columns to choose from were not as numerous as those for collaboration or personality. One could use the data from the post-programming task parts of the experiment, where the programmers themselves evaluated their own system in respects of how certain they were that what they had done was a good solution to the problem, whether or not they thought their solution was easily maintainable etc. However, we did not want to use these highly subjective measures. It was feasible that most of the pairs, regardless of the actual quality of the code they had made were quite pleased with their result, since they had decided to announce the task as done before filling out this form. Few pairs would quit the task when they felt it still needed a lot more work in order to be complete.

Since the pairs were supposed to program until they were done with the programming tasks of the experiment, the duration of the task was the primary measurement of performance in the experiment, and the "quality" of the solution was recorded only as a binary value of whether or not a pair's solution passed the test cases of the experiment. However, this binary value had some use, so it was used as one of the quality factors, along with duration for task 4. A combined "quality" score for all the three evaluated pair programming tasks (task 2, 3 and 4) was also a column available for use. Even though it was somewhat unsuitable, since the task analyzed and coded was task 4 only, this combined correctness score was included as a quality measure that could strengthen or weaken other results.

11.2. Relationships to be Investigated

In the following subchapters, the two relationships to be analyzed will be briefly described along with the second, part of the process of selecting suitable variables.

11.2.1. How Personality Influences Collaboration

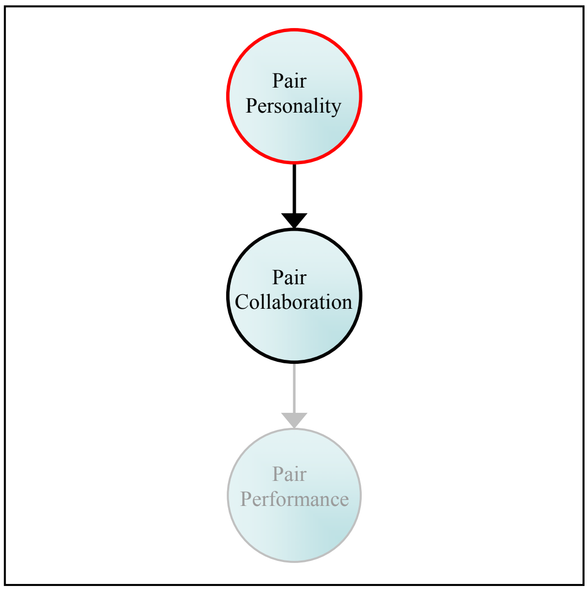

Figure 11: Does personality affect collaboration?

The primary focus of this thesis is to investigate the relationship between personality and collaboration, as shown in Figure 11. Personality factors for pairs were chosen as explanatory variables for the analyses of this relationship. As mentioned above, the personality factors chosen were mean and StdDev for every of the five Big Five traits. So, in total, there were ten explanatory variables. The *StdDev* personality column was for the analyses referred to as d*iff*, short for difference, for easier readability. Thus, "Extraversion mean", "Extraversion diff", "Agreeableness mean" and "Agreeableness diff" was the names of the first four of the ten personality variables we included in our analyses.

As response variables, nearly all collaboration categories were selected, and a statistical analysis was performed on these, one at a time.

The few collaboration categories that were not selected were those for which it would seem meaningless to make analyses. One of the categories omitted was "Not Applicable" from the *Result* category group. This category was meant to be used in normal N/A scenarios, that is, as a "filler" when it is not suitable to use any of the more meaningful categories *resolved* and *unresolved*.

The second category that was not included in the analyses was the *Task Focus* category "Unintelligible". Like the N/A category, "Unintelligible" is nothing more than a filler, and describes nothing more than the quality of the sound clip and the coder's ability to hear what is said.

The category hardest to argue for why we kept out of the analysis was the task focus "Compile and Test". However, it would be unsuitable to include this category, since we had decided to stop coding the task right before the last session of compiling and testing (not including it). Also, the collaboration during compiling and testing sequences is not an interesting focus for us, since it is an input/output heavy situation where the "communication" with the computer is the mean point. "Compile and Test" was rarely occurred until long clips of it at the very end of the task.

Propositions P_1-P_6 described in Chapter 12 below investigate this relationship between personality and collaboration.

11.2.2. How Collaboration Influences Performance

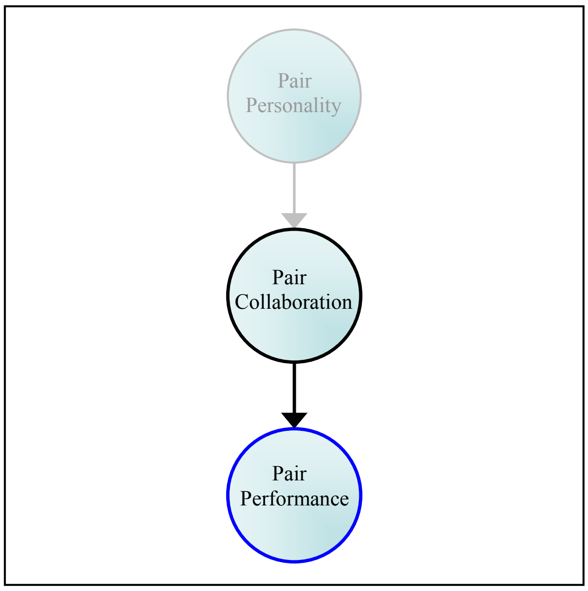

Figure 12: Does collaboration affect performance?

Although not my primary focus, I also investigated whether certain collaboration categories was more or less likely to lead to good performance in the pair programming experiment. Performance was, as stated above, measured by time use and correctness. As when investigating the relationship above, in Section 11.2.1, the three collaboration categories "Not Applicable", "Unintelligible" and "Test" were omitted from the analysis.

As one can see in Figure 12, the collaboration was the explanatory factor for this relationship, as opposed to in the focus described in the previous subchapter, where it was the response. Therefore, when investigating this relationship, the collaboration categories were all used as explanatory variables. As response variables, the three aforementioned performance measures were used; correctness for task 4, correctness for all tasks, and time used on the task.

Propositions P_7 and P_8 described in Chapter 12 below investigate this relationship between collaboration and pair programming performance.

12. Research Propositions

Based on a literature review which is described in the following chapters, I made a number of relationship propositions for personality impacts on collaboration and collaboration impact on performance. They are listed as P_1-P_8 below.

Each of the "sub-hypotheses" extracted from the literature were given a score based on the compatibility of the source to my problem statement, as well as how reasonable the sub-hypotheses seemed. The score was given from a scale from 1 to 4, where 1 meant that the sub-hypothesis was "very likely" to be true, and 4 meant that it was "slightly possible" that the sub-hypothesis would be confirmed by the analyses.

The best scoring ones among these short sub-hypotheses were then combined into a handful of well-defined propositions that could be the basis for direct analysis. These relationship propositions are:

P_1: Personality affects the use of certain collaboration categories.

P_2: Difference in personality increases the number of "communication transactions"; i.e. the collaboration categories that include high amounts of discussion will be used more often.

P_3: People with similar levels of extraversion will disrupt each other less.

P_4: Pairs consisting of two extroverts will discuss a lot regardless of type of discussion.

P_5: Agreeableness leads to more social/small talk.

P_6: Metacognitive statements are made more often by extraverted people.

P_7: The use of certain collaboration categories leads to better or worse pair programming performance.

P_8: The use of the interaction pattern *elaborative*, leads to better results.

12.1. Literature Based Proposition Suggestions

A lot of articles about personality and pair programming do not focus on collaboration and the articles about personality and collaboration are not about pair programming. However, a handful of articles were of great use to us, and they, along with the hypotheses one could deduct from them, are presented in the sections below (12.1.1 - 12.1.6).

The propositions are numerous and would if uncompressed form quite a long list. To compress them, I used a special format to describe them. A (-) in front of a dependent factor in this format indicates a negative correlation. For example (-) EC-f for the personality trait Extraversion diff, indicates that a high *Extraversion diff* (difference) leads to a low use of the collaboration category of flow (EC-f, see Chapter 8). The format is as follows:

# (Score). Factor

    Dependent factors. If multiple; divided by commas)

12.1.1. Sfetsos et al.

The article of Sfetsos et al. [Sfetsos 06] states that there is a significant correlation between mixed personalities and high amounts of *communication transactions*. They compare the mixed personality pairs to pairs with similar personalities. Based on this, we can make the following hypothesis:

1. Openness diff, Extraversion diff, Conscientiousness diff, Agreeableness diff, Emotional Stability diff
   (-)TF-PS, (-)TF-X, (-)IP-c, (-)IP-n, TF-D, TF-C, TF-P, IP-e, IP-r

12.1.2. Dick et al.

The article by Dick and Zarnett [Dick 02] is a primarily theoretical paper that claims that four traits are important for successful pair programming. These four are *communication*, *comfortable*, *confidence* and *compromise*. They explain these four traits in some detail. Communication is defined to be the ability to elaborate on ideas, but be able also not to say too much. The article does not mention personality types and how they might collaborate, but the article does suggest that people who elaborate much will perform pair programming well.

1. IP-e
   Pair programming performance (increased correctness or decreased duration).

12.1.3. Karn et al.

In 2005, Karn and Crowling wrote an article about personality types and software engineering collaboration with special focus on disruption. One of their findings, from an ethnographic study, is that in one of the groups they studied (which consisted of four developers) none of the members made any disruptions at all. They state that the fact that the group had one clearly dominant member, (the group's only extrovert), was the reason why they never disrupted each other. Furthermore, the authors claim that "some homogenous teams (particularly those dominated by INTs) run a real danger of falling into the no debate trap" [Karn 05]. Their results show that disruptions were good for performance of the teams. They do a new but similar investigation one year later [Karn 06]. In the latter, the 2006 study, the findings from the 2005 study, that disruptions benefit the performance, are directly contradicted. The view that disruptions are bad is also the most common view in the other sources. Because of the contradiction, no hypothesis will be made regarding the effect of disruption on performance.

2. Extraversion diff
   (-) EC-d

Since the article was about software engineering in general and not specifically pair programming, the hypotheses does not receive a score of 1.

Findings in this article also contribute by strengthening the hypothesis presented in the Sfetsos article above.

12.1.4. Williams et al.

The article by Williams, et al. [Williams 06] further suggests that different personalities were the best. They focus on the specific MBTI range of sensing vs. intuition. McRae and Costa's conversion sheet between MBTI and Big Five [McRae 89] indicates that sensing vs. intuition is highly correlated with the Big Five trait *Openness*. Since the article states that people of mixed scores on the sensing-intuition scale claim to feel more compatible with each other than those with more similar score on the scale, it is likely that the sensing-intuition different people collaborated fairly well also. However, the finding in the article are based on subjective checkboxes only where the subjects themselves stated how well they thought their partner during the pair programming matched them. Therefore, this sub-hypothesis will not be given a very high score.

3. Openness diff
   (-)TF-PS, (-)TF-X, (-)IP-c, (-)IP-n, TF-D, TF-C, TF-P, IP-e, IP-r.

12.1.5. Hannay et al.

In "Effects of Personality on Pair Programming" [Hannay 09a] there is a literature review that includes, among others, the aforementioned articles. In this review, the authors state that "None [of the articles] argued that homogeneous pairs would be better performers". This finding helps to strengthen the hypothesis from the Sfetsos article.

Based on the experiment described within, Hannay et al.'s article further states that certain factors are beneficial regarding pair programming performance. While it is undesired to translate this directly into theories about collaboration category use, one can make a weak hypothesis based on it.

3. (-)Extraversion diff, (-)Agreeableness diff
   (-)TF-PS, (-)TF-X, (-)IP-c, (-)IP-n, TF-D, TF-C, TF-P, IP-e, IP-r

12.1.6. Williams' Book

The book "Pair Programming Illuminated" [Williams 03] describes pairs of the three possible combinations of extraverts and introverts. They claim that extrovert-extrovert people will talk unnecessarily much, often about things outside the tasks, and thus spend a very long time on the tasks. However, the double extroverts will make a high quality code, due to their thorough discussions. Regarding extrovert-introvert, they present only hints for how pairs of these compositions should behave. The authors make no real indication (other than a joke) of how the collaboration might evolve. For introvert-introvert, the authors suggest that there will be less talk than for other extraversion combinations in pairs.

The book mentions what the authors refer to as the "professional driver problem", where one programmer continuously, and despite the navigators expressed or implied desires, is the driver. However, this problem will not be covered in my analyses, since in the experiment on which this thesis is based; the pairs were forced to change the roles at least once. Also, one would have to listen carefully for it in the audio in order to identify the problem. It might not be possible to identify it at all without video of the pair programmers.

The book also mentions another problem, in which one person thinks he is better than the other member of the pair, and thus treats this other member badly. This is mostly visible when he/she ignores the other one. This problem is thought to indicate that "the ignorer" has a low agreeableness score. Regrettably, both these interesting phenomena would be awkward to model using our categories and the big five personality system since we also decided to focus on the pairs as a whole, and not differentiate between the two individuals (see Section 7.4.3). The phenomena could, however, be interesting areas to investigate in future research, as described in Section 16.4.

2. Extraversion mean, (-)Extraversion diff
    -> TF-Z

2. (-)Extraversion mean, (-)Extraversion diff
    TF-PS, TF-X, IP-c, IP-n

12.2. Definition-Based Proposition Suggestions

A number of sub-hypotheses based only on the definitions of the big five traits and of our collaboration categories were also made. Since these were weighted less heavily in the process of generating research proposition, they are not discussed closely. The complete list of the definition-based proposition suggestions can be found in Appendix X6.

13. Analysis Description

The exact focus of this study was new and not much covered in available literature. Because of this, an exploratory analysis seemed the most appropriate. Inspired by the research on the direct link between personality and performance in [Hannay 09a], the statistics software jmp, version 7.0.2, by SAS was chosen as our tool for the analysis. Jmp has a mode of analysis called “partition” that is especially suitable for exploratory analyses. This mode is described in section 13.1 below.

Jmp also has a quite powerful support for scripts, and supports a rich scripting language that can be used in order to automate most parts of the software functionality. The script language is easy to use for people who are experienced with typical programming syntax such as loops and *if* statements. Scripts for all the analyses were made, and they were automated as much as possible within a reasonable time frame. I wanted to make the scripts so that they automated the analyses as much as possible, but I did not want to use more time on script making than the time I would save due to the automatic nature of the analyses.

The scripts would prove useful, since we changed our minds regarding what variables we included in the analyses several times (as seen above, in Section 11.1.2). The script could also be useful for potential future work on. For example could a future run with an extended sample or with different *split sizes* or *k-values* than before (both these concepts are described below) be based on these same scripts.

13.1. The Partition Platform

> The Partition platform recursively partitions data according to a relationship between the X and Y values, creating a tree of partitions. It finds a set of cuts or groupings of X values that best predict a Y value. It does this by exhaustively searching all possible cuts or groupings. These splits (or partitions) of the data are done recursively forming a tree of decision rules until the desired fit is reached.[JMP 07]

In jmp, a partition split analysis requires little effort by the user. One simply presses a button titled “split", and a *split* is made. The first split in a tree, the split which has the complete data sample as their basis, is referred to in this text as the *top split*. While splitting, a tree structure like the one in Figure 10 is generated.

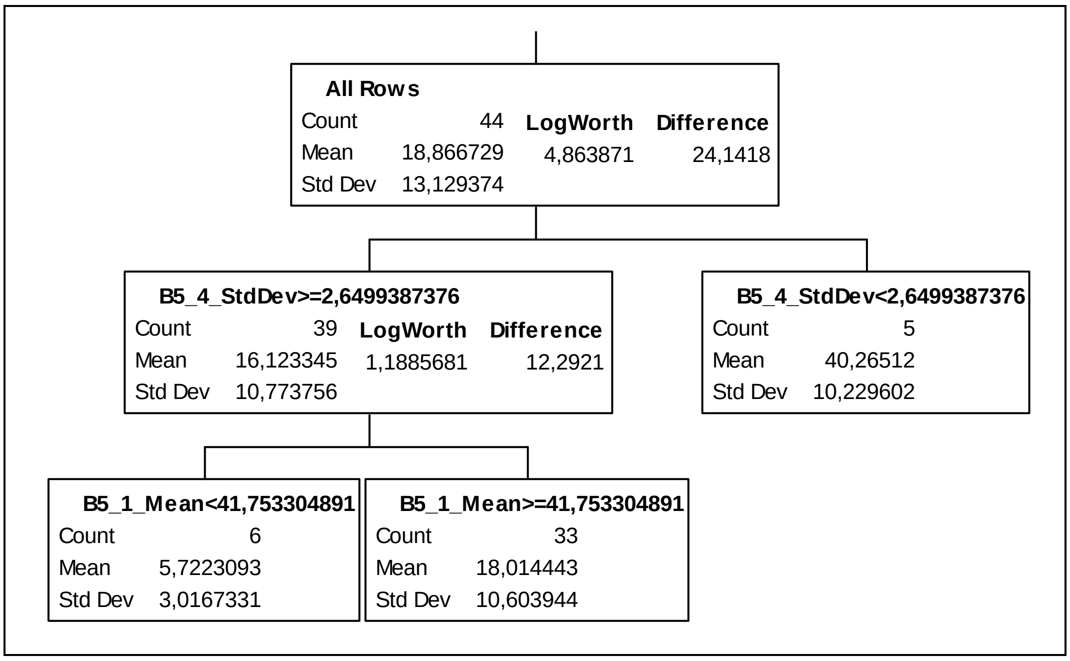

Figure 10: Partition: The first two splits with the Task Focus “Programming Aloud” as response variable, when measuring the relative amount of time the category occurs

According to convention, jmp was set up so that the significance of the split, the *LogWorth* (see Section 13.1.2.) was maximal. The splits are made by jmp, and ordered so that the rightmost of the two groups generated by the split has a higher mean value for the response variable than the leftmost group.

In Figure 10, we see that the rightmost group in the first split has a mean value of ≈40.27 in “Programming Aloud”. This means that for the pairs in this group, 40.27 percent of the time they used to solve tasks, were coded as the *Task Focus* “Programming Aloud”, while for the group at the left, only ≈16.12 percent of the time was coded with this category. This is quite a noticeable difference, and one can verify that the difference is significant by looking at the number in the LogWorth column in the “All Rows” box above the split. LogWorth is a measure of significance of the jmp splits, and is explained in detail in Section 13.1.2 below.

But simply knowing that there is a significant split is not enough. Which factor is affecting “Programming Aloud”, and in what direction? The headings of the two split groups contain *B5_4_StdDev*, which is the column name in the jmp data table for “the deviation between the two pair members in regards to emotional stability”, also known as *Emotional Stability diff* in this text. Since *B5_4_StdDev* is larger than 2.65 in the leftmost split box, and it is smaller than 2.65 in the right one, we know that a similar level (low difference) of emotional stability in the pairs leads to more time spent on “Programming Aloud”. We know this because the left group is stated to be high in *B5_4_StdDev*, and low in “Programming Aloud”, and oppositely for the right group, it is low in *B5_4_StdDev*, and high in “Programming Aloud”.

As stated earlier, the group that has the highest mean value in the response variable is always placed at the right side of the split diagram. Therefore, it is easy to read and understand the split diagrams quickly and accurately when one is used to them. One can simply look at the inequality symbol (< or >), read the name of the variable that is used in the split, and control the LogWorth, in order to know a great deal about the split.

The example in Figure 10 has a second split under the leftmost group of the first split. One can see that the *BF_1_Mean*, the mean value of the two pair members' extraversion score, is the splitting factor. In contrast to the first split, for this second split, the group with the highest score on the personality factor is the one on the right. This means that a high combined score in extraversion for a pair's two members leads to more "Programming Aloud". However, for this split we see that the significance, the LogWorth, is lower, only 1.18, which is slightly below what is required based on the choices we have made. Thus, this split is not identified as significant. These significance decisions are discussed below, in Section 13.1.2.

13.1.1. Minimum Size Split

In jmp, an important setting for the partition platform is "minimum size split". The number selected here decides how small a split group can be. Indirectly, the minimum split size will thus decide the number of splits it is possible to perform.

Since the minimum size split in the example in Figure 10 above was set to 5, it would be possible to split this tree further. The box in the middle at the bottom has a count greater or equal to the double of the split size (5*2=10), so it could be split further. At least one of the new groups formed by a split of this box could also be used for additional splits.

As stated in Section 9.1, the sample size in our case was 44. Thus, a minimum split size of 23 would mean that it would not be possible to make any splits regardless of the input material. Two split groups of minimum 23 in size would require at least 46 observations. A minimum split size of 22 would work, but it would limit the number of splits to one, as further splits of the two partitions (which would inevitably be of size 22 exactly), would be impossible with a split size that high.

The default value for minimum split size in jmp is 5, which is regarded as sufficient for diminishing the effect of outliers; individual values that fall outside the overall pattern [Moore 06]. We don't want the minimum split size so low that potential measurement errors or other oddities would often be placed in a split group by themselves. For example, if one of the subjects of the experiments had read the instructions for the personality test wrong, and had answered oppositely of what he really meant for every single question, this person's personality profile would probably stand out as an outlier that could disturb linearity of the sample.

In the earlier analysis based on the same experiment, the focus was to find a direct link from personality to performance. For this, a split size of 10 was used [Hannay 09a]. The authors quote [Briand 01] in their choice of this number and state that it is chosen to avoid outliers. The article they quote [Briand 01] states that a split size of 10 "should be sufficiently large to capture significant trends."

There does not seem to be a universal consensus on a suitable split size. Few available articles deal with the subject. A minimum split size of 10% is used in [Alkushi 05]. In the case of the analyses presented in this thesis, where the sample size was 44, 10% would be rounded down to 4; not that different from jmp's default suggestion of 5.

Because of the meager amount of material one could find about split size decisions and argumentations, it was difficult to decide on a split size. In the first two rounds of analysis, split sizes of both 5 and 10 were used, and the results for both split sizes noted individually.

After noticing that the results were mostly the same when using the split size of 5 and 10, i.e. the same variables was chosen by jmp with similar results, a split size of 5 only was used during the last analysis rounds. It seemed more reasonable to choose this smaller split size considering the somewhat small sample size of the material.

13.1.2. LogWorth and p-value

In jmp, LogWorth is a measurement of significance used to describe the "quality" of a split. In [Hannay 09a], it is described like this, with a reference to a technical report made by SAS, the jmp developers [Sall 02] at the end:

> The "LogWorth" index in each parent node is a significance measure of the difference in mean values for the observations in each child node with regards to the dependent variable. Specifically, LogWorth = $-\log_{10}(p)$, where p is the adjusted probability of the observed data under the hypothesis of the means being equal. […]. The adjusted p-value takes into account the number of different ways splits can occur. It is fair compared to the unadjusted p-value and to the Bonferroni p-value.

In statistics, the p-value is a significance measure, meaning that it shows how statistically significant a result is. "The probability, computed assuming that H_0 is true, that the test statistic would take a value as extreme or more extreme than that actually observed is called the **P-value** of the test. The smaller the P-value, the stronger the evidence against H_0 provided by the data" [Moore 06]. One can think of the p-value as the probability of a finding being a coincidence. With a p-value of 0.05, we are 95% (1-p) sure that the result is not coincidental.

The conventional p-value of $p<0.05$ is a suitable value for this analysis. Since personality and collaboration are the most important factors we measure, we should avoid the most restrictive values commonly used, such as 0.01 and 0.005. While these lower values will be suitable for making strong assumptions on the most tangible of things, such as production numbers in a factory, or cyclical natural phenomena, they will probably be too restrictive for us, since collaboration and personality are complex factors which are likely to have quite a large variance. Also, since the sample size is rather small, the variance will be slightly high. Therefore, we would most likely not find very many usable results if we used a very low p-value as the significance threshold.

By choosing 0.05 as the significance level, a LogWorth of at least 1.30 was required for a split to be noted as significant, and used to argue for or against hypotheses, since $-\log(0.05) \approx 1.30$.

13.1.3. K-Fold Cross-Validation

> Cross-validation, sometimes called rotation estimation, is the statistical practice of partitioning a sample of data into subsets such that the analysis is initially performed on a single subset, while the other subset(s) are retained for subsequent use in confirming and validating the initial analysis.[Wikipedia 09]
>
> K-Fold cross-validation is a technique that is suited for smaller data sets. In this type of cross-validation, the data are partitioned randomly into a number of groups that you specify (K) and the following procedure is followed: […] The model is fit to the complete data set, […] after iterating through the K starting values, the CVRsquare is set to be the average of the R2 values for the K iterations. […] for smaller data sets the CVRsquares should be less variable and much more reliable than they would be for the corresponding (1-1/K)*100% Holdback cross-validation fit statistics.[JMP 07]

Since the sample size was rather small, a k-fold cross-validation seemed suitable. A k of 44 (which is also the sample size) was selected. This would make the cross-validation results predictable and stable and not based on random selection as the case would have been if we had selected a smaller k for the k-fold. The jmp software is fast, so the cross-validation was performed nearly instantaneously. To include a 44-fold-cross-validation did not lead to noticeable increases in the time it took to open a new partition window and perform the partitioning.

The cross-validation has an R^2 score as output. This value signifies the percentage of variance in the data that the model (the split tree) accounts for. Cross-validation scores were not used for anything in particular during my analyses, but they are listed in the table showing the table analysis of proposition $P_{2\ \mathrm{to}}$; Table 8 in Section 14.1.2.

13.2. Analysis Procedure

As primary analysis procedure, instead of simply splitting the data and noting the significant findings, a slightly different approach was used. Whenever a top split (first split) was significant, the explanatory variable that the split was based on was removed and the analysis was run again. This was done because if a split on agreeableness would lead to a 3.01 LogWorth for the split, and a split on conscientiousness would lead to a 2.84 LogWorth (this one also very high), only the fist one, agreeableness, would appear as the first split if one was running the analysis only once. The conscientiousness split would probably not appear further down in this potential fully expanded tree where agreeableness was the first split either, since the agreeableness-based first split would most likely alter the material, so that conscientiousness was no longer a very significant split factor for any of the two groups formed by the split.

However, if agreeableness was removed from the list of explanatory variables and the analysis run again, conscientiousness would in fact appear as the first split. This procedure was repeated until there was no longer a significant split in the first split of the tree. Thus, for response variables where the first split when all explanatory variables were included was not significant, no additional analysis runs were needed.

The procedure described above was chosen since it focuses mainly on top splits, which are the most proper splits to consider when one is describing general trends. When considering top splits only, the findings regard the whole data set and not only a subset of it. Also, when focusing on significant splits only, we have a well-defined place to stop the analysis, which would otherwise, because of its exploratory nature not “end” naturally until every single possibility was tried.

This procedure will never block out any significant top splits due to its stop definition: The splits are made by jmp on the basis of maximizing their significance. This means that the top split with all independent variables included will always be more significant than the top split of the next run, where the variable that was in the top split last time was removed. When the top split is no longer significant, we can with confidence stop the process, since the next top split (after removing yet one variable) will be **even less** significant.

On the other hand, this process will exclude potential significant splits further down the tree than the top one, but this exclusion is a choice we made. It did not seem suitable, in this particular setting, to compare a split based on one particular data set (that was generated by a split above) with a split based on a completely different data set (generated by a different split).

In addition to the regular analysis procedure described in this section, the complete split trees were investigated for the variables involved in the propositions, in order to look for trends and patterns in the trees. Often, this extra check lead to the realization that certain findings was unsuitable to base statements on, since it was common that one or more of their split trees were internally contradicting of themselves. For proposition P_2, this additional procedure was expanded even further. This will be explained in details in the results section for this particular proposition, section 14.1.2.

The relevant factors only were included as variables for the investigation of most of the specific propositions. For example, for a hypothesis like “Metacognitive statements are made more often by extraverted people”, the category “Metacognitive” and the personality trait “Extraversion” was the only ones included.

14. Analysis Results

Our analysis did not confirm most of the propositions: For all the rather specific propositions, no evidence in support of them was found. However, all the major, broader propositions were found to be apt.

The first one of these confirmed propositions was P_1 which was a general proposition regarding the complete set of possible relationships between personality and collaboration. If P_1 had been rejected, it would mean that there was no correlation between these two factors at all, and to look into the other propositions would be a waste of time.

The second of the important and confirmed propositions: P_2, was a broad and important proposition, and was the one most closely based on the literature. It was primarily inspired by the MBTI based pair programming and collaboration experiment by Sfetsos et al. [Sfetsos 06], and strengthened by statements from other research papers [Karn 05] [Hannay 09a]. The results showed a noticeable trend in support of P_2.

The third relationship proposition which turned out to be supported by the analyses was P_7, a proposition closely related to P_1, but where the other relationship, namely the one between collaboration and performance was the focus, instead of personality and collaboration as in P1-P6. Significant results were found for this hypothesis as well.

For the other five relationship propositions, no results could confirm them, and they had to be rejected.

From the results, we know that personality does influence collaboration, and collaboration does influence results, but not in all the ways that were expected based on literature review and analysis of the definitions of the categories and the personality traits. Table 6 sums up the propositions and the reason why they were rejected or not. Below it, a closer argumentation for each of the propositions, as well as the results regarding them is listed.

P	Confirmed?	Reason
P_1	Yes	Several collaboration categories have a highly significant correlation with personality factors.
P_2	Yes	Several collaboration categories that describe the “communication transaction heavy” situations are significantly affected by personality differences.
P_3	No	For both correctness and task duration, elaborative has no significant impact
P_4	No	No significant impact found.
P_5	No	Several collaboration categories with much discussion are affected by extraversion, and mostly in one direction, but the impacts are not significant.
P_6	No	No significant impact found.
P_7	Yes	Several collaboration categories have a highly significant correlation with performance measures.
P_8	No	No significant impact found.

Table 6: Summary of the main hypotheses and the analysis results

14.1.1. P_1: Does Personality Affect Collaboration?

Personality affects the use of certain collaboration categories.

Table 7 lists all the significant findings when I used the procedure described in Section 13.2. The *negative* column indicates whether the collaboration category is affected positively or negatively by the personality factor. For example, the very first row shows a correlation between "Extraversion mean" and "TF O" (Which is the *Task Focus* category "Other Relevant Tasks", see Chapter 8). Since we see in the third column, that it is indicated that the correlation is negative (√), we know that if there is a high mean Extraversion score for the pair, their collaboration will consist of less "TF O" than for the other pairs.

P_1 is supported. As shown in Table 7, when running the regular analysis procedure nearly all personality factors significantly influenced the amount of one or more of the collaboration categories, both when measuring time use for each category, (left part of Table 7), and when measuring the amount of times the categories occurred (the right half of Table 7).

Personality factor	**% time**	**Neg-ative**	**Log Worth**	***p*-value**	**% clips**	**Neg-ative**	**Log Worth**	***p*-value**
Extraversion mean	*TF-O*	√	1.43	0.04	*EC-f*	√	1.48	0.03
	CL-P		1.31	0.05	*EC-d*		1.48	0.03
					IP-n	√	1.37	0.04
Extraversion diff	*TF-Z*		3.94	0.00	*IP-x*		2.00	0.01
	IP-x		1.51	0.03				
Agreeableness mean	*IP-s*		2.48	0.00	*TF-P*	√	2.07	0.01
	BC-q	√	1.87	0.01	*TF-C*		2.07	0.01
	IP-x		1.65	0.02	*Re-r*	√	2.00	0.01
	Re-r	√	1.50	0.03	*IP-q*	√	1.97	0.01
	CL-D	√	1.47	0.03	*Re-u*		1.88	0.01
					CL-M	√	1.35	0.04
Agreeableness diff	*TF-X*		1.35	0.04	*Re-r*	√	1.47	0.03
Conscientiousness mean	*TF-D*	√	2.16	0.01				
Conscientiousness diff								
Emotional Stability mean	*TF-Z*	√	1.39	0.04				
	CL-M		1.59	0.03				
Emotional Stability diff	*TF-PA*	√	4.86	0.00				
Openness mean	*BC-i*		3.70	0.00	*BC-i*		4.68	0.00
Openness diff	*IP-r*		2.11	0.01	*IP-r*		1.60	0.03

Table 7: Influence of personality on time used on collaboration categories

14.1.2. P_2: Will Different Personality Pairs Communicate More?

Difference in personality increases the number of "communication transactions"; i.e. the collaboration categories that include high amounts of discussion will be used more often.

The proposition is supported. "Communication Transaction Heavy" collaboration refers to the collaboration categories *Off Task* (Z), *Elaborative* (e), *Responsive* (r), *Unresolved* (u), *Cross Purpose* (x) and *Disruption* (d). These are the categories for which we - for better or worse - can expect to observe much discussion or dialog. For some of them, the large amount of dialog is included in the clips that are coded with this category (example: elaborative). For others, the category implies much discussion in the clip right after the occurrence of the category (example: disruption). Several of these collaboration categories were found to be significantly positively affected by variability in personality, which speaks in favor of P_2.

Another group of categories describe situations of little communication, namely *Programming Silently* (PS), *Silence* (X), *Nonresponsive* (n), *Consensual* (c), *Stonewalling* (s), *Flow* (f) *and Resolved* (r). These categories were found to be applied less often to clips during the coding of the personality-different pairs' protocols compared to when coding the dialog of the pairs with a personality more similar to each other. This too strengthens P_2.

The results for some categories contradicted these findings, but mostly, the findings had non-significant LogWorth-values. The only significant finding that spoke against P_2 can be seen in the left part of Table 7 above. For pairs with a high variability in Agreeableness (*Agreeableness diff*), the programmers are silent (TF-X) for longer periods than the pairs with two similar people in them are. A more detailed look at these primary analysis results for P_2 is shown in Appendix X5.

For P_2, in addition to the regular top split analysis described in Section 13.2, I did a larger combined analysis similar to that in [Hannay 09a]. For this, the complete split trees were investigated, and the order of the splits as well as an indication of whether they were significant or not was noted. A selection of categories was picked; the ones that indicated especially much or especially little discussion, as described above. These were divided into the two groups Communication-intensive Categories (C) and Silent Categories (S). The categories included in C were the same as the ones selected as *Communication Transaction Heavy* categories above. The categories in S were the other group mentioned above.

I then performed the complete tree analyses on these categories. This analysis can be seen in Table 8. The numbers indicate how early the split occurred (with 1 being the first split of the tree, 2 the second and so on). A '*' indicates a significant split. The +/- in front of the numbers indicate whether the split showed a positive or a negative correlation. If -, use of the dependent factor was decreased when scores on the independent factor (in this case: variability between the two pair members for some personality trait) was high. At the bottom of each column, the n-fold cross validation (explained in 13.1.3) is given, and at the very bottom, the overall R^2, which indicates the ratio of explained variance.

	Communication-intensive Categories (C)										
	TF-Z	***IP-e***		***IP-r***		***Re-u***		***IP-x***		***EC-d***	
	Time	**Time**	**Clips**	**Time**	**Clips**	**Time**	**Clips**	**Time**	**Clips**	**Time**	**Clips**
Extraversion diff	+1* -5 +6	+4*			+5	+1 -2 +4	+1 -2 +4	+1* -4	+1*	-6	-3 +4
Agreeableness diff		-1	-3 -6	+6		+5	+5	+2	+2		+5
Conscientiousness diff	+3* +4	-5	-5	+5	+6	-3	-3				
Emotional Stability diff	+2* -7	-3*	-2*	+2	+1*				+3 -4	-2 -3 +4 -5*	-2
Openness diff		-2	-1 +4	+1* +3 -4	+2 -3 +4			+3		+1	+1 +6
n-Folded R^2	.36	.46	.16	.32	.24	.12	.16	-.04	.04	-.04	-.04
Overall R^2	.56	.59	.37	.50	.48	.33	.36	.21	.25	.26	.30

	Silent Categories (S)											
	TF-PS	***TF-X***	***IP-n***		***IP-c***		***IP-s***		***EC-f***		***Re-r***	
	Time	**Time**	**Time**	**Clips**	**Time**	**Clips**	**Time**	**Clips**	**Time**	**Clips**	**Time**	**Clips**
Extraversion diff				-1*	-3 +6	+5	+2* +3	+2 +4	+6	+3 -4	+4	+5
Agreeableness diff	+2	+1*	+4 -5	+2* -3 +4			+1 -4	+1 -3		-5	-1* +3	-1* +3
Conscientiousness diff	+1 -4 +5	+2 +5		-6 -7	-2	-1 -2 -6					+5	
Emotional Stability diff	+3	-3 +4 +6	+1 -2*		-1 +4 +5				+2 +3 -4 +5*	+2		
Openness diff	+6		+3 -6	+5		+3 -4			-1	-1 -6	-2	-2 +4
n-Folded R^2	.28	.32	.12	.40	.16	.20	-.09	-.14	-.04	-.04	.21	.21
Overall R^2	.49	.54	.36	.59	.40	.44	.24	.21	.26	.30	.40	.40

Table 8: Secondary exploratory analysis for P_2

To summarize the findings, the procedure from [Hannay 09a] was used:

> [The summary table] ranks each independent variable according to how early associated splits occurred, by the formula max splits + 1 - split number. In our case, max splits was seven, that is, no trees had more than seven splits. Thus, if, say, Agreeableness assumed a significant second split in a model, then that split would contribute 7+1-2=6 to the ranking for Agreeableness. The '*'-columns only count significant splits.

Coincidentally, in the split trees for my new analysis, the maximum number of splits that occurred was 7, as in the example below. Thus, the very same formula in its original, unaltered form could be used. My summary is shown in Table 9.

	C*-S*	C-S	C*	C	S*	S
Extraversion diff	26	6	25	30	-1	24
Agreeableness diff	1	-5	0	12	-1	17
Conscientiousness diff	5	4	5	-2	0	-6
Emotional Stability diff	2	-31	-1	-8	-3	23
Openness diff	7	40	7	25	0	-15

Table 9: Secondary exploratory analysis for P_2 aggregated

The first two columns are the results when summing up splits found in the split trees from the *Communication-intensive Categories* (C) and then subtracting the splits from the *Silent Categories* (S). This resulting number could indicate the overall influence that the personality trait has on the amount of Communication-intensive collaboration. A near zero score here would indicate that P_2 is not appropriate, since it would mean that a high difference in personality will not influence the amount of communication-intensive collaboration, or equally influence both communication-intensive and silent categories. A high positive score, would be in favor or P_2, and would indicate that difference in personality traits leads to an increase in communication-intensive Collaboration. A negative score would contradict P_2.

Since there are 12 factors included in the *Silent Categories*, and only 11 in the *Communication-intensive Categories*, the results will be slightly less positive than they could have been if some silent factors were removed to make it equal. Since it would be hard to argue for removal for a single category; why it was chosen instead of others, the choice was made that it was best to keep this inequality.

The four rightmost columns list the results when measuring *Communication-intensive Categories* (C) and *Silent Categories* (S) individually. Asterisks (*) signify that only significant splits are included in the column.

As one sees from Table 9, there is indeed a quite high number of splits that indicates that variability in personality has an impact on the amount of *Communication Transactions / Communication-intensive collaboration*. This is most evident for extraversion, and it confirms our findings during the regular analysis; that P_2 is a valid proposition, since it seems that the amount of communication-intensive Collaboration do increase when the pairs have different personalities. But watch out for differences in emotional stability; it could possibly lead to a *de*crease, as we see in the C-S field, where the score is -31!

14.1.3. P_3: Does Similar Levels of Extroversion Lead to Less Disruption?

People with similar levels of extraversion will disrupt each other less.

We find no support for the proposition, so it must be rejected. Pairs with a similar level of extraversion are not found to disrupt each other more, but there is not significant evidence to support that they will disrupt each other less either. The partition trees involved in the analysis of this proposition contradict themselves quite a lot, with signs of both positive and negative correlations at different split levels. However the top splits of the trees are the ones that are most significant in this case and they all point to a positive correlation between extroversion differences and the disruption category.

As seen in Table 7, the category IP-x (Cross-purpose), which is a category not that dissimilar to disruptions, will increase when *extraversion diff* increases. This finding strengthens the belief that the less extraversion different people will be likely to disrupt each other more rather than less.

Because of the contradicting nature of the split trees, and the lack of significant findings, we must conclude that extraversion difference in the pairs will not affect the use of the disruption category. However, further investigation of this proposition could be interesting for a different or expanded study.

14.1.4. P_4: Will Two Extroverts Discuss a Lot?

Pairs consisting of two extroverts will discuss a lot regardless of type of discussion.

The proposition is not supported. Several collaboration categories that describe situations of much discussion are affected by extraversion, and mostly in one direction, but the impacts are far from significant on all categories. The only significant finding was that pairs with a high mean score on extroversion will disrupt each other more often than their introvert counterparts. All in all, the evidence is too weak to be of any support of P_4.

14.1.5. P_5: Do Two Agreeable People Small Talk Much?

Agreeableness leads to more social/small talk.

Proposition rejected. When measuring time used on the categories; neither for *Off-Task*, *Metacognitive* nor *Other Relevant Tasks*, the three categories deemed as the ones involving small talk, will the agreeableness have any significant effect. Actually, for all these three categories, the non-significant findings suggest that agreeableness would rather *de*crease than increase their occurrence, which contradicts our hypothesis. All top splits indicate this contradiction, but the trees are internally contradicting themselves at splits further down the line, so their results should not be taken too seriously.

When measuring the amount of clips that was coded as the categories, *Metacognitive* is in the top split found to be significantly less frequently seen in the collaboration of high agreeableness pairs. Regarding amount of time the pairs used on *Metacognitive* discussions, the first split indicates a similar correlation; that the highly agreeable people have less of it, but this time; the finding is slightly insignificant.

So if anything, highly agreeable people small talk less than their not-so-agreeable counterparts in a pair programming situation; a fact that would surprise many if it could be proven. We can with some confidence state that agreeable people will use the cognitive level of *Metacognitive* less.

14.1.6. P_6: Are Metacognitive Statements Made More by Extraverts?

Metacognitive statements are made more often by extraverted people.

P_6 is not supported by our evidence. The significance of the splits in the analysis of this proposition is incredibly low, and the split trees are contradicting themselves internally. The results show no reason to believe that extroversion has any effect on the occurrence use of the "Cognitive Level" of *Metacognitive*.

14.1.7. P_7: Does Collaboration Affect Performance?

The use of certain collaboration categories leads to better or worse pair programming performance.

This proposition is supported. As shown in Table 10, when running the regular analysis procedure (described in Section 13.2 above); both correctness for task 4, and the duration for task 4 were significantly affected by one or more collaboration categories. The results when measuring the total time of the clips that were coded as a given category is shown in the left part of Table 10, and the results when measuring the amount of clips that were coded as the categories is shown in the right half of Table 10.

Performance category	% time	Neg-ative?	Log Worth	p-value	% clips	Neg-ative?	Log Worth	p-value
Correctness (o4_correct)	*TF-P*	√	2.65	0.00				
Duration (Task4_length)	*Re-U*		5.15	0.00	*Re-u*		10.10	0.00
	EC-d		4.70	0.00	*EC-d*		4.70	0.00
	EC-f	√	4.70	0.00	*EC-f*	√	4.70	0.00
	TF-PA		3.16	0.00	*Re-r*	√	2.85	0.00
	TF-D	√	2.93	0.00	*IP-x*		2.36	0.00
	IP-x		2.36	0.00				
	Re-r	√	2.25	0.01				
	TF-O	√	1.87	0.01				
	IP-s		1.49	0.03				

Table 10: Category occurrence effects

The performance measure o2o3o4_correct (the combined scores for all the three pair programming tasks), did not yield any significant results when used as dependent variable.

14.1.8. P_8: Is *Elaborative* Good for the Performance?

The use of the interaction pattern "elaborative", leads to better results.

The proposition is rejected. *Elaborative* has a non-significant influence on both correctness and duration. When measuring correctness, the partition trees consist of nothing but insignificant splits which frequently contradicts themselves internally. Positive correlations are about as frequent as negative ones. Even though the two most significant splits both point to *elaborative* being beneficial for correctness, they are both far from our desired significance level, and are medium level splits in the trees.

For duration, the situation is similar. The splits are all far from significant, and they indicate both positive and negative correlations. However, the top splits, which are also the most significant splits for these variables, both indicate that *elaborative* leads to a shorter duration.

What we know from this is that the "interaction pattern" *elaborative* does **not increase** the task duration, even though the category is an intrinsically discussion rich "interaction pattern". One could think that *elaborative* would increase the duration, since it involves much time spent on discussion, rather than programming actions, but this turns out not to be the case. Thus, *elaborative* is definitely not "bad" for the performance.

Because of the conflicting results and non-significance of the findings here, we cannot declare that P_8 is true. However, P_8 is not completely disproved, so a closer look at this hypothesis in another setting, or with a larger sample size could be interesting at a later time.

14.2. Other Findings

In addition to the analyses in regards of the hypothesis, a partition tree analysis of nearly every single category (with exceptions described in Section 11.2) was run in jmp to see what personality factors influenced them. In all tables below, non-significant results, based on the decision to use 0.05 as the confidence level limit, are listed in gray. A negative influence means, as in the tables above, that there is less of the second factor if there is more of the first.

14.2.1. Complete Investigation of Personality Factors' Influence on Collaboration

As described in Section 11.2.1, the first (and main) step was to find out whether or not the personality profiles of the pairs had an impact on the occurrence of certain collaboration categories in this pair's collaboration.

Nearly every category was investigated by using the regular analysis procedure described in Section 13.2. The results are listed in Tables 11-16 below.

Since the occurrence data was comparing interaction sequences only, most of the Task Foci was not analyzed for occurrence. These areas are grayed out in the table below.

Time used on Task Focus	Time use is affected by	Neg-ative?	Log Worth	p-value	Occurrence is affected by	Rev-erse?	Log Worth	p-value
X – Silence	Agr diff		1.35	0.04				
	Con diff		1.11	0.08				
O – Other relevant	Ext mean	√	1.43	0.04				
	Emo diff		1.15	0.07				
PS – Prog. Silently								
PA – Prog. Aloud	Emo diff	√	4.86	0.00				
	Ext mean		1.26	0.05				
P – Programming					Agr mean	√	2.07	0.01
C – Comprehension	Ext mean	√	1.05	0.09	Agr mean		2.07	0.01
D – Task Description	Con mean	√	2.16	0.01				
	Emo diff		1.24	0.06				
Z – Off task	Ext diff		3.94	0.00				
	Emo mean	√	1.39	0.04				

Table 11: Personality influences on Task Foci

Begin Characteristics	Time use is affected by	Neg-ative?	Log Worth	p-value	Occurrence is affected by	Rev-erse?	Log Worth	p-value
i – Imperative	Ope mean		3.70	0.00	Ope mean		4.68	0.00
s – Suggestion								
a – Assertion	Ext mean		1.86	0.01				
q – Question	Agr mean	√	1.87	0.01	Agr mean	√	1.97	0.01

Table 12: Personality influences on Begin Characteristics

Interaction Pattern	Time use is affected by	Neg-ative?	Log Worth	p-value	Occurrence is affected by	Rev-erse?	Log Worth	p-value
n – Nonresponsive					Ext mean	√	1.37	0.04
e – Elaborative	Agr diff	√	1.23	0.06				
r – Responsive	Ope diff		2.11	0.01	Ope diff		1.60	0.03
x – Cross-purpose	Agr mean		1.65	0.02	Ext diff		2.00	0.01
	Ext diff		1.51	0.03				
s – Stonewalling	Agr mean		2.48	0.00	Agr mean		1.25	0.06
	Agr diff		1.28	0.05				
c – Consensual	Emo diff	√	1.11	0.08	Con diff	√	1.11	0.08

Table 13: Personality influences on Interaction Patterns

End Characteristics	Time use is affected by	Neg-ative?	Log Worth	p-value	Occurrence is affected by	Rev-erse?	Log Worth	p-value
f – Flow	Ext mean	√	1.09	0.08	Ext mean	√	1.48	0.03
					Emo mean		1.11	0.08
d – Disruption	Ext mean		1.09	0.08	Ext mean		1.48	0.03
					Con mean	√	1.11	0.08

Table 14: Personality influences on End Characteristics

Result	Time use is affected by	Neg-ative?	Log Worth	p-value	Occurrence is affected by	Rev-erse?	Log Worth	p-value
u – Unresolved	Agr mean		1.28	0.05	Agr mean		1.88	0.01
					Ext diff		1.16	0.07
r – Resolved	Agr mean	√	1.50	0.03	Agr mean	√	2.00	0.01
	Agr diff	√	1.27	0.05	Agr diff	√	1.47	0.03
					Ope diff	√	1.08	0.08

Table 15: Personality influences on Results

Cognitive Level	Time use is affected by	Neg-ative?	Log Worth	p-value	Occurrence is affected by	Rev-erse?	Log Worth	p-value
M – Metacognitive	Emo mean		1.59	0.03	Agr mean	√	1.35	0.04
	Agr mean	√	1.01	0.10				
D – Domain level	Agr mean	√	1.47	0.03	Agr mean	√	1.17	0.07
	Ope mean	√	1.04	0.09				
S – System level					Agr mean		1.02	0.10
P – Program level	Ext mean		1.31	0.05				

Table 16: Personality influences on Cognitive Levels

14.2.2. Alternative Formatting for the Collaboration Influence on Performance Table

Shown below is Table 17, which is similar to Table 10 in Section 14.1.7 above, but ordered by category instead of LogWorth/significance. Although this table provides no new information, the alternative formatting will be more readable for people trying to find relationships from collaboration to performance, while the table shown in 14.1.7 was better to illustrate that there are indeed significant influences. Instead of being divided into a left and a right part; one for time use and one for occurrence, as the earlier tables are; this table is divided into a top and a bottom part. This was done mainly since it seemed esthetically better for this particular table.

Personality factor Time use	Affects	Neg- ative?	Log Worth	p- Value
TF - Program. Aloud	Task duration		3.16	0.00
TF - Task Descript.	Task duration	√	2.93	0.00
TF - Programming	Correctness	√	2.65	0.00
TF - Other rel. tasks	Task duration	√	1.87	0.01
IP - Cross-Purpose	Task duration		2.36	0.00
IP - Stonewalling	Task duration		1.49	0.03
EC - Disruption	Task duration		4.70	0.00
EC - Flow	Task duration	√	4.70	0.00
Re - Unresolved	Task duration		5.15	0.00
Re - Resolved	Task duration	√	2.25	0.01
Personality factor Occurrence				
IP - Cross-Purpose	Task duration		2.36	0.00
EC - Disruption	Task duration		4.70	0.00
EC - Flow	Task duration	√	4.70	0.00
Re - Unresolved	Task duration		10.10	0.00
Re - Resolved	Task duration	√	2.85	0.00

Table 17: Personality influences on Cognitive Levels

14.2.3. Transitive Deductions

By investigating the personality factors that influence category use, and the categories that influence performance, one can make assumptions about possible relationships between personality and performance, via collaboration. These deductions are shown in Tables 18a and 18b below. Table 18a lists the transitive findings when measuring the duration of the clips that were coded as the categories. Table 18b lists the results when measuring the number of clips that was coded as the categories.

The results we see here should not be taken very seriously due to the questionable way in which they are formed. However, it is an interesting attempt to see if it is possible to trace a personality influence all the way "through" collaboration and that it will in turn affect performance. If nothing else; the results here can be used as hypotheses for further investigation.

High score on trait	Leads to	Of category	Which makes the duration		Hence: high score on trait	Makes duration
Extraversion mean	Less	Other relevant tasks	Shorter		Extraversion mean	Longer
Extraversion diff	More	Cross-purpose	Longer		Extraversion diff	Longer
Agreeableness mean	More	Cross-purpose	Longer		Agreeableness mean	Longer
Agreeableness mean	More	Stonewalling	Longer		Agreeableness mean	Longer
Agreeableness mean	Less	Resolved	Shorter		Agreeableness mean	Longer
Conscientiousness mean	Less	Task Description	Shorter		Conscientiousness mean	Longer
Emotional stability diff	Less	Programming Aloud	Longer		Emotional stability diff	Shorter

Table 18a: Deductions about how personality affects performance when using time use of collaboration categories as mediating factor

High score on trait	Leads to	Of category	Which makes the duration	Hence: high score on trait	Makes duration
Agreeableness mean	More	Unresolved	Longer	Agreeableness mean	Longer
Extraversion mean	More	Disruption	Longer	Extraversion mean	Longer
Extraversion mean	Less	Flow	Shorter	Extraversion mean	Longer
Agreeableness mean	Less	Resolved	Shorter	Agreeableness mean	Longer
Extraversion diff	More	Cross-purpose	Longer	Extraversion diff	Longer

Table 18b: Deductions about how personality affects performance when using the frequency of use of the collaboration categories as mediating factor

As one can see, high scores of nearly all personality traits are shown to lead to an undesired impact on duration; i.e. it will be longer. Extraversion and agreeableness occurs in several of the transitive relationships, and might therefore be seen as "worst". A possible explanation for the negative representation of extraversion and agreeableness here might be that extroverts and agreeable people will possibly small talk a lot, which consumes time. However, we have seen from the rejection of propositions P_4 and P_5, which investigated just this, that this is not the case. Therefore, why there is possibly a longer task solving duration when the pair members are high in extraversion or agreeableness remains unexplained.

For conscientiousness, one could assume that their focus on "getting it right" and attention to details might lead to a lengthier pair programming session, with more use of the "Task Foci" *Programming Aloud* and *Programming*, *disruptions*, *unresolved* interaction patterns and *Program Level* thinking. However, there is no evidence of any of this in the analysis, where conscientiousness was found to have nothing else than a negative impact on the use of the "Task Focus" of *Task Description*.

The only personality factor that is shown here to be beneficial if quick programming is the goal is the deviation in the pairs' emotional stability score. This finding is more likely than not coincidental. However, a very high difference in it would mean that one of the pair members has a quite high score on the trait. It has been shown, among other places in [Barrick 01], that emotional stability is beneficial for all kinds of work. For this reason, it is not totally unthinkable that a difference in this personality trait will be good for pair programming performance, but the difference between them would in such a case be a confounding factor, lurking behind the fact that the pair has one very emotional stable member, which might be the main contributor, and the reason for the good performance.

End

After having analyzed the collaboration involved in pair programming, we can answer the overall research question *”Does personality affect the collaboration of pair programmers?”* It is clear from the analysis results that personality does affect collaboration, and in some aspects; in a much larger degree than personality will affect the performance of pair programming. A brief summary of the thesis can be found in the beginning, in the introduction text on page 11.

In the following chapters, some of the possible threats to validity in this thesis work are described, as well as proposals for future direction of work on the same subject.

15. Possible Validity Threats

There are two areas that could cause potential validity threats in the work described in this text. They are investigated in a section each below.

One should also mention the validity threats of the actual pair programming experiment [Arisholm 07]. Since I was no part of it myself, I obviously had no influence in either the design or execution of it. The pair programming experiment is referred to as the authors as a quasi-experiment. It has a number of potential validity threats. However, the article describing the experiment has a very thorough description of the threats and the measures the authors did in order to avoid problems caused by the validity threats.

The main threat, and the reason why the study is referred to as a quasi-experiment, is the selection issue for people that were selected to be in the control group consisting of individual programmers. All data about the individual programmers were gathered in 2001, while for the pair programmers, the data were gathered in 2004-2005. A number of factors could cause the latter group to perform better, and it is stated in the text that they did indeed perform better. A possible reason is that newer and better tools and versions of software were used by the latter group.

The authors adjust the results of the control group by using complex models, which the authors demonstrate will diminish the problem of this performance gap. In any case, since these differences are between the groups of "individual programmers" and "pair programmers" only, it will not be relevant for this thesis, since my analyses include the pair programmers only.

The other notable validity threat is that it was the project leaders in the companies that contributed subjects to the experiment who chose which of their fellow employees that would attend the experiment. The project leaders were also the ones who assigned the subjects into expertise groups (junior, intermediate, senior).

One might speculate that some companies did not enforce a random selection of people, but chose those people that the company, rather than the experiment, would benefit the most out of sending, since the companies got the same compensation regardless on outcome of the experiment. Some companies might have chosen their very best developers, in order to try to make their company look better. Others might have chosen among the developers which were least essential to the company's daily development, since these programmers were the ones they could do without with the least amount of programming power loss for the company. However, this turned out not to be the case, however. The amounts of juniors, intermediates and seniors were fairy similar.

Since the subjects were assigned to expertise groups in a rather subjective way, one could suspect that some people did not really belong to the expertise group in which they were placed. However, the authors of the article that describes the experiment have made sure that the expertise groups are apt:

> Nevertheless, as is evident from [tables in the article], seniors worked faster (with more correct task solutions) than did intermediates, who, in turn, worked faster (with more correct task solutions) than did juniors. [Arisholm 07]

Also, since this thesis does not focus on programmer expertise, but personality, even if there had been problems with the programmer expertise concept, it would not have influenced the results herein.

15.1. Categories

15.1.1. Constructs

Regarding construct validity, since the coding schema is an abstract system, it is possible that the categories are "wrong" or overly subjective, and that they do not represent what they should, or that they will not be understood similarly by different people.

The coding schema reliability check, which turned out to lead to satisfactory results, is a way to argue that our constructs was not a substantial threat to the validity of the study. Also, the categories were, with only a few exceptions, based closely on previous schemas and literature and thus tested and described, sometimes quite thoroughly, even before we included them into our coding schema.

15.1.2. Content

Regarding content validity, it is possible that there are some flaws. The categories were made after listening through audio files. We then tested each version of the schema and refined it until we were satisfied. The categories were, as mentioned, heavily based on the related literature that we found when we searched the literature databases. Never during my audio file coding sessions did I think a category was "missing" in the final schema, and only rarely was I in doubt on which category to choose. None of the other coders complained about lacking categories in the final schema either.

If we wanted to extend the focus to additional factors of collaboration, those new category dimensions could easily have been added to the schema, but for the six dimensions in our final schema, the collection of categories appear as complete. The category definitions are meant to make the borders between the different categories as apparent as possible.

15.2. Analysis

15.2.1. Selection

As seen in section 9.1, most British and medium-expertise expertise pairs were excluded from the analyses. This might have influenced the results in some degree. Since a coding of the remaining material could be done at a later time, a new analysis based on the complete data set coded would avoid this risk.

15.2.2. Confounding Variables

Since we involve only the personality traits as the explanatory factors in the analyses on influences on collaboration, there is a risk that one or more factors might be additional influencing variables. Since "Task Complexity" and "Programmer Expertise" were found to be two quite important predictors in the previous research based on the experiment [Hannay 09a], these two factors are especially likely candidates as possible confounds in the analyses in this thesis.

However, the focus of this thesis was to investigate the possible relationships between personality and collaboration. A separate investigation is being done into the relationship between programmer expertise and collaboration, and it is likely that an investigation will be done some time in the future about task complexity and collaboration. When all these three, (plus maybe some additional factors) are ready, one can attempt to make a combined analysis on all these factors to see how they influence collaboration as a whole, and how much each factor influences it.

16. Possible Future Directions

Even though the results I found confirmed the main research propositions and were statistically significant, collaboration and pair programming is (as shown in earlier chapters) an area of research that is open for exploration, and very little research similar to this exists. By using one or several of the suggestions listed below, one could likely discover more about the relationship.

16.1. Larger Sample

Because of time constraints, we did not code and analyze the complete set of the 98 programming pairs' sound recordings. As stated in Section 9.1, a total of 47 audio files were coded, but three of these turned out to erroneous and were discarded. Thus, the final sample size for my analyses was 44, which was somewhat less than half of the total audio files available.

Variation typically decreases when the sample size is increased. With a decreased variation, many findings would likely be shown as more significant. Also, other findings that were not found at all in my analyses might emerge, since the measures of significance depend on, among other things, the variation of the data material.

The protocol coding parser (described in Section 9.4) is compatible for any number of pairs, the personality data for the rest of the pairs are already gathered (they were gathered during the experiment), and the analysis procedures are saved in scripts. Therefore, an expansion of this research would consist simply of coding more audio files, and running and summarizing the analyses.

16.2. Analysis with Other Personality Measures or Methods

The analyses both in this thesis and in [Hannay 09a] were done with each personality factor as separate variables. Some kind of combination of factors could lead to interesting findings. Or one could select only the people with a high or a low difference in each trait and do the analyses on these similar or dissimilar minded pairs only.

One could also try to look into the individual personality trait scores of the people in the pairs. However, since the focus was on collaboration in this case, it might (as discussed earlier) not be apt to have this kind of individualistic focus.

To use another analysis method than exploratory split tree analysis is also possible, even though our method was argued to be the most suitable for this particular setting. One could try a more "powerful" automatic analysis, like the one in the confirmatory analysis of [Hannay 09a] or something completely different. A lot of possibilities exist.

Another interesting area of research would be to investigate whether pairs that were more likely to collaborate in a certain way would be more or less likely to like pair programming and feel that it was a comfortable and effective technique. Because of the results from [Arisholm 07] and other literature, we know that pair programming is rarely positive for employers in terms of cost, but many could choose to encourage its use regardless of this, due to other benefits of pair programming. Among these are the social factors of collaborating closely, which could make some employees happier and more relaxed and better performers overall. If we find out more about which people are more likely to like pair programming best, one could know for whom it could be an appreciated break from their individual programming routines, and for whom it would only be an annoying chore that would “break their flow”.

16.3. Long-term Pairing

If two people work together for a long time, will they get better? In the first analyses done on the pair programming experiment [Arisholm 07] they state the following:

> A majority of the subjects had little or no experience with PP before the experiment and, in most cases, had not pair programmed with their assigned partners before. Consequently, the results of this study might be a quite conservative measure of the effects of PP, since the pairs had probably not reached their maximum level of combined efficiency during the experiment: Anecdotal evidence suggests that it takes developers from a few hours to a few days to make the transition from individual programming to efficient pair programming. […] Future experiments should thus include a mix of pairs with different degrees of pair cohesiveness.

It is very likely that the collaboration would be quite different as well in pairs with more pair programming experience. But will personality matter more or less in such a case? One could guess at both. Introverts and other people who might initially be uncomfortable with the collaborative nature of pair programming, might “ease into it” after a while and learn to like it. On the other hand, dominant people might get even more dominant over time. A third possibility is that different-minded pairs (as seen in proposition P_2 in the analyses) over time will learn to utilize their differences to a maximum effect over time, and that they will be even better collaborators, and maybe performers, after having pair programmed a while.

Many new interesting findings could emerge if one looked at more long-term pair programming effects. However, a new experiment would have to be performed in order to investigate this, and a very costly one at that, since the experiment would need to be conducted over a period of at least several days. One could suggest to include only highly experienced pair programmers in this potential new experiment, and to compare the results from it to those from the experiment we already have. However, this would reintroduce the validity threat from the experiment (that the groups were investigated at different times), and it might not be ideal. It would undoubtedly be the quickest and less expensive way to investigate this effect, though.

16.4. Individual Focus

In our coding and analysis process, we focused on pairs as a whole, as described in Section 7.4.3.

Data about who of the two pair members the initiator of the dialog in each clip was, and who of the two that was the driver and navigator could be interesting for analyses later. It might be especially interesting in relation to my focus area, personality. Maybe the more extroverted people are more likely to initiate a discussion? Will an individual with a high score on agreeableness be more likely to answer fully and not ignore the other person? If employing an individual focus, one could also investigate the two additional phenomena described by Williams (discussed in section 12.1.6).

However, focusing on individuals during protocol coding and analysis would be very time consuming, compared to when treating the pairs as single entities.

16.5. Distributed Pair Programming

Pair programming is called remote or distributed pair programming when the two pair members are collaborating similarly to when doing regular pair programming but when the two are separated geographically. They must still work at the same time, and communicate continuously by some means. Often, voice communication and some shared environment is used, as seen in [Natsu 03], but distributed pair programming systems can also attempt to simulate regular pair programming more accurately, by including some sort of video conferencing. This last idea could for example be a semi-transparent view of the other pair member, like in the suggestion presented in [Stotts 04].

Since distributed pair programming is, as the name suggests; distributed, it is more flexible; people will not have to meet physically in order to pair program. Distributed pair programming could be used for tasks not directly related to programming, but where pair programming-like techniques are suitable. For example a design or requirement meeting, could be done by a representative of the customer of a system, and a representative from the software firm, without the need of what might be a long and/or time-consuming distance to travel. If they used a distributed pair programming setup, they could suggest things and discuss, and come up with a version they both were comfortable with in collaboration.

This distributed variant of pair programming is not included as in any of the experiments reviewed here that investigate personality and pair programming. It is feasible that the personality impacts on distributed pair programming are quite different from those that appear when performing regular pair programming, since the distributed pair programming is less personal, and more computer based. It is also highly likely that the nature of the collaboration is quite different when performing distributed pair programming compared to when pair programming on the same physical machine.

16.6. A Closer Look at the Transitive Relationships

If the analyses are run again, for example with a larger sample, one should take a more serious look into the possible transitive relationships between personality and performance, with collaboration as mediating factor. Possible relationships are illustrated by using an informal deduction using the logic rule of transitivity in Section 14.2.3, but it could be done more scientifically by using different methods. A suggestion from the supervisor is to model it by using a mediator variable in a path analysis instead.

This transitive relationship is not the focus of this thesis. However, it would be among the things that employers and process improvers would be most interested in, since they could look for the “good” personality traits in their employees and combine them into pairs “correctly” for ideal effect. For this reason, the personality-collaboration-performance transitivity is an important area of further investigation.

Acknowledgements

I wish to thank the following people for their help:

Jo Hannay – my supervisor on the masters thesis. A better supervisor would be hard to imagine. He consistently went above and beyond “the calls of duty”, and for any kind of question, any day of the week, he would help. I thank you for all the guidance, help, contributions and support throughout my thesis work.

Vigdis By Kampenes. Thanks for the help with, among other things, coding schema development.

Bente Nordbø. Thanks for your efforts in the collaboration on making the coding schemas and your contributions during the audio file coding. Thanks also for driving me to Simula each time we went there.

Stian Friberg and Alexander Ottesen. Thanks for your great work with coding the audio files.

Martha Kutscha. Thanks for finding the start times for task 4 in the audio files. It saved us quite some time and prevented mistakes.

Jo Hannay again, and Erik Arisholm, Tore Dybå, Hans Gallis, Dag Sjøberg, Harald Engvik and the other people who administered, executed, documented and performed the first rounds of analyses on the large pair programming experiment. Without your efforts and ideas, this master thesis would not have existed.

The 196 it consultants that were subjects in the experiment. Thanks for participating. Your occasional jokes while you solved the tasks made the coding process more interesting and less monotonous.

References

[Alkushi 05] A. Alkushi, Z.H. Abdul-Rahman, P. Lim, M. Schulzer, A. Coldman, S. E. Kalloger, D. Miller, C. B. Gilks, "Description Of A Novel System For Grading Of Endometrial Carcinoma And Comparison With Existing Grading Systems," *American Journal Of Surgical Pathology*, vol. 29, no. 3, pp. 295-304, Mar 2005

[Arisholm 07] E. Arisholm, H. Gallis, T. Dybå, D.I.K Sjøberg, "Evaluating Pair Programming With Respect To System Complexity And Programmer Expertise", *IEEE Transactions On Software Engineering*, vol. 33, no. 2, pp. 65-86, Feb 2007

[Bakeman 97] R. Bakeman, J. M. Gottman, *Observing Interaction,* Second Edition. Cambridge University Press. Cambridge, MA, USA, 1997

[Barrick 01] M. R. Barrick, M. K Mount, T. A. Judge, "Personality And Performance At The Beginning Of The New Millennium: What Do We Know And Where Do We Go Next?" *International Journal Of Selection And Assessment*, vol. 9, no. 1-2, pp. 9-30, Mar-Jun 2001

[Briand 01] L. C. Briand, J. Wüst, "Modeling Development Effort In Object-Oriented Systems Using Design Properties," *IEEE Transactions On Software Engineering*, vol. 27, no. 11, pp. 963-986, 2001

[Bryant 06] S. Bryant, P. Romero, B. Du Boulay, "The Collaborative Nature Of Pair Programming," *Extreme Programming And Agile Processes In Software Engineering, Proceedings*, vol. 4044, pp. 53-64, 2006

[Bryant 04] S. Bryant, "Double Trouble: Mixing Qualitative And Quantitative Methods In The Study Of Extreme Programmers," *Proceedings Of The 2004 Ieee Symposium On Visual Languages - Human Centric Computing,* pp: 55–61, 2004

[Cao 05] L. Cao And P. Xu, "Activity Patterns Of Pair Programming", *Proceedings Of The Proceedings Of The 38th Annual Hawaii International Conference On System Sciences (Hicss'05) Track 3*, vol. 3, pp. 88 Published: 2005

[Chan 01] C. K. K. Chan, "Peer Collaboration And Discourse Patterns In Learning From Incompatible Information," *Instructional Science*, vol. 29, no. 6, pp. 443-479, 2001

[Chao 06] J. Chao, G. Atli, "Critical Personality Traits In Successful Pair Programming," *Agile 2006, Proceedings*, pp. 89-93, 2006

[Chi 97] M. T. H. Chi, "Quantifying Qualitative Analyses Of Verbal Data: A Practical Guide," *Journal Of The Learning Sciences*, vol. 6, no. 3, pp. 271-315, 1997

[Choi 08] K. S. Choi, F. P. Deek, I. Im, "Exploring The Underlying Aspects Of Pair Programming: The Impact Of Personality," *Information And Software Technology*, vol. 50, no. 11, pp. 1114-1126, Oct 2008

[Cloninger 04] S.C. Cloninger, *Theories Of Personality : Understanding Persons*, 4th Ed. Pearson/Prentice Hall. Upper Saddle River, NJ, USA, 2004

[Dick 02] A. J. Dick, B. Zarnett, “Paired Programming And Personality Traits,” *Third International Conference On Extreme Programming And Agile Processes In Software Engineering*, pp. 82-85, 2002

[Dybå 07] T. Dybå, E. Arisholm, D. I. K. Sjøberg, J. E. Hannay, F. Shull, ”Are Two Heads Better Than One? On The Effectives Of Pair Programming,” *Ieee Software*, vol. 24, pp. 12-15, 2007

[Ericsson 93] K. A. Ericsson, H. A. Simon, *Protocol Analysis: Verbal Reports As Data Revised Edition.* The MIT Press. Cambridge, MA, USA, 1993

[Freudenberg 07] S. Freudenberg, P. Romero, B. Du Boulay, “'Talking The Talk': Is Intermediate-Level Conversation The Key To The Pair Programming Success Story?” *Agile 2007, Proceedings*, pp. 84-91, 2007

[Goldberg 93] L. R. Goldberg, “The Structure Of Phenotypic Personality-Traits”, *American Psychologist*, vol. 48, no. 1, pp. 26-34, Jan 1993

[Hanks 06] B. Hanks, “Student Attitudes Toward Pair Programming,” *Iticse '06: Proceedings Of The 11th Annual Sigcse Conference On Innovation And Technology In Computer Science Education*, pp. 113-117, 2006

[Hannay 09a] J. E. Hannay, E. Arisholm, H. Engvik, D. I. K. Sjøberg, “Personality and Pair programming,” to appear in *IEEE Transactions on Software Engineering.* 2009

[Hannay 09b] J. E. Hannay, T. Dybå, E. Arisholm, D. I. K. Sjøberg, “The Effectiveness of Pair Programming: A Meta-Analysis,” to appear in *Information & Software Technology*, 2009

[Hogan 00] K. Hogan, B. K. Nastasi, M. Pressley, “Discourse Patterns And Collaborative Scientific Reasoning In Peer And Teacher-Guided Discussions,” *Cognition And Instruction*, vol. 17, no. 4, pp. 379-432, 1999

[Hughes 03] J. Hughes, S. Parkes, “Trends In The Use Of Verbal Protocol Analysis In Software Engineering Research,” *Behaviour & Information Technology*, vol. 22, no. 2, pp. 127-140, Mar-Apr 2003

[Jmp 07] *JMP Statistics And Graphics Guide*, Release 7,: Sas Institute Inc., Cary, NC, USA, 2007

[Karn 05] J. S. Karn, A. J. Cowling, “A Study Of The Effect Of Disruptions On The Performance Of Software Engineering Teams,” *2005 International Symposium On Empirical Software Engineering (Isese), Proceedings*, pp. 403-411, 2005

[Karn 06] J. S. Karn, A. J. Cowling, “A Follow Up Study Of The Effect Of Personality On The Performance Of Software Engineering Teams,” *Proceedings Of The 2006 Acm/Ieee International Symposium On Empirical Software Engineering*, pp. 232-241, 2006

[Katira 04] N. Katira, L. Williams, E. Wiebe, C. Miller, S. Balik, E. Gehringer, “On Understanding Compatibility Of Student Pair Programmers,” *Technical Symposium On Computer Science Education,* pp. 7-11, 2004

[Keirsey 08] Keirsey.com, *About 4 Temperaments*. Available at `http://keirsey.com/handler.aspx?s=keirsey&f=fourtemps&tab=2&c=inspector`, `http://keirsey.com/handler.aspx?s=keirsey&f=fourtemps&tab=5&c=mastermind` (Retrieved January 1st, 2008)

[Layman 06] L. Layman, “Changing Students, Perceptions: An Analysis Of The Supplementary Benefits Of Collaborative Software Development,“ *19th Conference On Software Engineering Education & Training, Proceedings*, pp. 159-166, 2006

[Lim 97] K. H. Lim, L. M. Ward, I. Benbasat, “An Empirical Study Of Computer System Learning: Comparison Of Co-Discovery And Self-Discovery Methods,” *Information Systems Research*, vol. 8, no. 3, pp. 254-272, Sep 1997

[Mayrhauser 99] A. Von Mayrhauser, S. Lang, “A Coding Scheme To Support Systematic Analysis Of Software Comprehension,” *IEEE Transactions On Software Engineering*, vol. 25, no. 4, pp. 526-540, Jul-Aug 1999

[Mcrae 89] R. R. Mcrae, P. T. Costa, “Reinterpreting The Myers-Briggs Type Indicator From The Perspective Of The 5-Factor Model Of Personality,” *Journal Of Personality*, vol. 57, no. 1, pp. 17-40, Mar 1989

[Moore 06] D. S. Moore, G. P. McCabe, *Introduction To The Practice Of Statistics*, 5th Ed. W. H. Freeman And Company. New York, NY, USA, 2006

[Myers 97] M. D. Myers, *Qualitative Research In Information Systems,* The University Of Auckland, New Zealand. Available at `http://www.qual.auckland.ac.nz` (Retrieved January 15th, 2008)

[Natsu 03] H. Natsu, J. Favela, A. L. Moran, D. Decouchant, A. M. Martinez-Enriquez, “Distributed Pair Programming On The Web,” *Proceedings Of The Fourth Mexican International Conference On Computer Science (Enc 2003)*, pp. 81-88, 2003

[Okada 97] T. Okada, H. A. Simon, “Collaborative Discovery In A Scientific Domain,“ *Cognitive Science*, vol. 21, no. 2, pp. 109-146, Apr-Jun 1997

[Olson 92] G. M. Olson, J. S. Olson, M. R. Carter, M. Storrosten, “Small Group Design Meetings: An Analysis Of Collaboration,” *Human-Computer Interaction*, vol. 7, no 4, pp 347-374, 1992

[IFI 08] IFI, *How Do Pair Programmers Collaborate?* Available at `http://www.ifi.uio.no/forskning/grupper/isu/php/masters/m23.php` (Retrieved June 6 Th, 2008)

[Pervin 01] L. A. Pervin, O. P. John, *Personality: Theory And Research*, 8th Ed. Wiley. NY, USA, 2001

[Raad 00] B. De Raad, *The Big Five Personality Factors : The Psycholexical Approach To Personality.* Hogrefe & Huber. Seattle, WA, USA, 2000

[Sall 02] J. Sall, *Monte Carlo Calibration Of Distributions Of Partition Statistics* Available at `http://www.jmp.com/software/whitepapers/pdfs/montecarlocal.pdf`, Nov 2002

[Sfetsos 06] P. Sfetsos, I. Stamelos, L. Angelis, I. Deligiannis, "Investigating The Impact Of Personality Types On Communication And Collaboration-Viability In Pair Programming - An Empirical Study," *Extreme Programming And Agile Processes In Software Engineering*, pp. 43-52, 2006

[Shneiderman 80] B. Shneiderman, *Software Psychology: Human Factors In Computer And Information Systems.* Winthrop Publishers, Inc. Cambridge, MA, USA, 1980

[Smith 89] D. C. Smith, "The Personality Of The Systems Analyst: An Investigation," *ACM SIGCPR Computer Personnel*, vol. 12, pp. 12-144, 1989

[Stotts 04] D. Stotts, J. M. Smith, K. Gyllstrom, "Support For Distributed Pair Programming In The Transparent Video Facetop," *Extreme Programming And Agile Methods - Xp/ Agile Universe 2004, Proceedings*, vol. 3134, pp. 92-104, 2004

[Thomas 03] L. Thomas, M. Ratcliffe, A. Robertson, "Code Warriors And Code-A-Phobes: A Study In Attitude And Pair Programming," *SIGCSE Bull*, vol. 35, no. 1, pp. 363-367, 2003

[Transana 08] Transana Website, Wisconsin Center For Education Research, University Of Wisconsin-Madison, Madison, WI, USA. Available at `http://www.transana.org/` (Retrieved November 13th, 2008)

[Visram 04] K. Visram, "Extreme Programming: Pair-Programmers, Team Players Or Future Leaders?" *IASTED Conf. On Software Engineering And Applications*, pp. 659-664, 2004

[Wikipedia 09] Wikipedia Contributors, *Cross-Validation,* Wikipedia, The Free Encyclopedia. Available at `http://en.wikipedia.org/w/index.php?title=Cross-validation&oldid=251150112` (Retrieved February 15th, 2009).

[Williams 06] L. Williams, L. Layman, J. Osborne, N. Katira, "Examining The Compatibility Of Student Pair Programmers," *Agile 2006, Proceedings*, pp. 411-420, 2006

[Williams 03] L. Williams, R. Kessler, *Pair Programming Illuminated.* Addison-Wesley. Boston, MA, USA, 2003

Appendices

X1. Coding Schemas

X1.1. Combined Categories

Other

- Ad-hoc-work - Working without discussing the long-term solution
- Break
- Private discussion - Things outside of the experiment are discussed.
- Disagreement that remains unsolved - A disagreement that ends with silence or inconclusive answer, or the actual statement ”let’s move on” or similar, when it is clear that there is not an agreement.

Good

- Discussion with agreement - An open discussion that is non-hostile where they reach an agreement (compromise, fair persuasion etc.). It can evolve into a ”disagreement”.
- Suggestion accepted - A suggestion receives a positive feedback.
- Planning of future work - Long-term discussion about long-term plans for the solution.
- Programming, duo - Both participants collaborate closely about code extension and are continuously discussing what should be done next and what is done now.
- Question answered - A question is answered, and the reply is understood/appreciated by the question asker.
- Disagreement solved - It is a disagreement from the beginning, but they agree after discussing it.

Bad

- One-sided break - One participant takes a break.
- Suggestion ignored - A suggestion from one is ignored by the other in some way.
- ”Override” - One person forces his opinion through.
- Passive person - One of the two is not involved in the development/discussion, due to lack of knowledge, energy or other reasons
- Programming, solo - One person is writing code, while the other is either silent or unenthusiastically consensual
- Question ignored - One person asks a question, but it is not answered, or answered so poorly that is does not benefit the questioner.
- Outside work - One of the people does private tasks during the pair programming session.

X1.2. Improved Categories

Other

- Other work - Small tasks relating to the experiment, but with no relevance to cooperation. Picking up printouts, asking the experimenter about the rules of the experiment etc.
- Break - Both participants taking a break. Physically absent from the workspace. No work-related activities in this period
- Private discussion - Things outside of the experiment are discussed. Not a break. Used only when what is discussed has no relevance at all to the work. For example, the phrase "Do you have much experience with Java?" must be coded as "planning of future work".
- Disagreement that remains unsolved - A disagreement that ends with silence or inconclusive answer, or the actual statement: "let's move on" or similar, when it is clear that there is not an agreement.

Good

- Suggestion accepted - A suggestion received a positive feedback.
- Planning of future work - Long-term discussion about long-term plans for the solution. Preferably with a 50-50 contribution rate to the discussion by the two participants, but a ratio as uneven as 80-20 is acceptable.
- Programming, duo - Both participants collaborate closely about code extension and are continuously discussing what should be done next and what is done now. Actual programming (keyboard use directed towards the code) must be happening.
- Question answered - A question is answered, and the reply is understood/appreciated by the question asker.
- Disagreement solved - It is a disagreement from the beginning, but they agree after discussing it.

Bad

- One-sided break - One participant takes a break, while the other continues the work, either by programming a bit alone or studying the problem (reading the code, the task description) or does other assignment related small tasks.
- Suggestion ignored - A suggestion from one is ignored by the other in some way.
- ”Override” - One person forces his opinion through.
- Passive person - One of the two is not involved in the development/discussion, due to lack of knowledge, energy or other reasons
- Programming, solo - One person uses the keyboard and edits the code. At least one participant is silent. A possibility is that both are silent, and all one can hear is an occasional ”mm-hmm”. It could also be that one person comments what is being done, while the other is silent or ignores what is being said.
- Question ignored - One person asks a question, but it is not answered, or answered so poorly that is does not benefit the questioner.
- Outside work - One of the people does private tasks during the pair programming session.

X1.2.1. Explanation for Start and Stop of Clips

Other

- Other work - Starts when it is obvious that the work is not about programming. Stops when new clip begins.
- Break - Starts when it is clear that both members are taking a break. Stops when they resume work.
- Private discussion - Starts when the discussion is not about the task. Stops when new clip begins.
- Disagreement that remains unsolved - Starts when the disagreement begins. Ends when the “agree to disagree” or similar, or when a new clip begins.

Good

- Suggestion accepted - Starts when a new suggestion is made. Ends when the discussion of the suggestion is done.
- Planning of future work - Starts when the discussion is obviously about planning off future work. Stops when this planning is done.
- Programming, duo - Starts when programming (keyboard use) is initiated and both people contribute to the discussion. Ends when no keyboard use is longer audible and the discussion is no longer about the previous programming. The usage of the category implies that there is audible keyboard use.
- Question answered - Starts when a new question is asked. Ends when the discussion about the question is done.
- Disagreement solved - Starts when a conflict appears. Ends when it is clear that the disagreement is resolved. There is now a common understanding of the subject in which there was earlier a disagreement.

Bad

- One-sided break - Starts when it is obvious that one person is taking a break. Stops either when also the other person takes a break (pause), or when both are back to work.
- Suggestion ignored - Starts when a suggestion is made. Stops when new clip starts
- Question ignored - Starts when a question is asked Stops when new clip starts
- ”Override” - Starts at first sign of dominant behavior (ignoring the other’s comments or suggestions, aggressive persuasion attempts, often initiated by an interruption. Ends when new clip starts.
- Passive person - Start right before a significant period where at least one person does not contribute actively to the collaboration. Ends when collaboration resumes as two-sided dialog.
- Programming, solo - Starts when programming (working on the keyboard) starts (but the other person is not contributing to the collaboration). Ends when programming is obviously over (more than 10 seconds (starts a clip of ”passive person” etc.) or when the other person starts contributing actively. The usage of the category implies that there is audible keyboard use.
- Outside work; Starts when it is obvious that one person does private tasks. Stops when a new clip begins.

X1.3. Expanded Categories

Choose one category of the ones in X1.3, plus one of the following four categories (which are divided into two groups):

Abstraction level - High

- The totality of the system – “What happens if I do this?" Connections between components; methods and classes. Discussions about a high-level ”snapshot” of the program.

- Real world - Link between the real world and the task. "OK. I’m going to buy coffee. Then I’ll usually put on the money first, and then select the type of coffee I want. We should have it like that in our solution too.”

Abstraction level - Low

- Syntax discussion - "No, you have to use *long* there, not *int*.", "I spot a typo over here”.

- Variables, methods, etc. - The most abstract of the low abstraction level discussions. Here, the dialog is at method level, but still code focused and does not require a good mental mode. “I guess it’s getSugar we have to use here”.

X2 Reliability Check Calculation Rules

We will count every clip in the file, and then do an arithmetic mean of the scores of them all.

Examples:

* ParfrP	-> PacfrP	= 5/6
* CarfrS	-> PaefrP	= 3/6
* ParfrP	-> CqnduS	= 0
Total: 8/18		= 4/9

X2.1. Score Calculations for the Categories:

- U is not counted.
- T is not counted. (Since one was supposed to stop coding before the last test).
- Silence vs. everything else is 0 score. It is considered to be impossible to misunderstand X. There must have been a mistake if x is used erroneously.
- O vs. Z -> 3/6 score. Since the border between them is sometimes somewhat thin. At their max, however, these categories are quite different (discussion about yesterday's party vs. problems with Eclipse).
- PA vs. PS -> 2/6 score. Programming is going on, but the dialog form in these two is totally dissimilar.
- PA vs. long interaction sequence categories
    - P(rogramming) gives 1/6, C(comprehension) gives 0
    - a(ssertion), q(uestion) and s(uggestion) all give 1/6
    - n(onresponsive) and c(onsensual) give 1/6, the rest: 0.
    - f(low) gives 1/6, the rest: 0.
    - resolved / unresolved is irrelevant, and give 0
    - P(rogram level) gives 1/6

  Example1: PanfrP / PA -> 5/6
  Example2: CqedrS / PA ->0

- PS vs. long categories: 0. PS is silent, so it must be 0. (Except vs. X: 4/6)
- D vs. long interaction sequence categories:
    - P(rogramming) gives 0, C(comprehension) gives 1/6
    - All begin characteristics give 1/6
    - All interaction patterns give 0.
    - f(low) gives 1/6, others 0.
    - Resolved / unresolved is irrelevant, and give 0
    - P(rogram level) gives 0/6, S(ystem) gives 1/6.
- All other categories in Task Focus vs. long interaction sequence, lead to a score of 0.

X3. Coding Schema Comparison Table

Nordbø, Walle	**Gary M. Olson, Judith S. Olson, Mark R. Carter, and Marianne Storrbsten**	**Takeshi Okada & Herbert A. Simon**	**Kai H. Lim, Lawrence M. Ward, Izak Benbasat**	**Kathleen Hogan, Bonnie K. Nastasi, Michael Pressley**
Collaboration in Pair Programming	*Small Group Design Meetings: An Analysis of Collaboration*	*Collaborative Discovery Scientific Domain*	*An Empirical Study of Computer System Learning: Comparison of Co-Discovery and Self-Discovery Methods*	*Discourse Patterns and Collaborative Scientific Reasoning in Peer and Teacher-Guided Discussions*
Other work	Project Management Other Goal			
Break				
Private discussion	Digression			Digressions
Disagreement that remains unsolved Disagreement solved		(Disagreement) to a hypothesis. Suspending conclusion. Argument about justification.		
Suggestion ignored Suggestion accepted	Alternative Criterion	Agreement to the hypothesis. Hypothesis Alternative hypothesis Extension of the hypothesis. Justification through experimental results. Justification using several experimental results.		Presents query Reacts (agrees, neutral) Reacts (disagrees) Presents (idea, partial idea, information, summary)
Planning of future work	Issue Meeting Management Summary Walkthrough	Plan for new experiments to test hypotheses. Summory of data Description of results Testability.	Formulating strategy/action	
Programming, duo Programming, solo			Describing physical action Reading screen output	Responsive Elaborative
Question answered Question ignored	Clarification	Requests for explanation.	Seeking Understanding Clarifying Interpreting output	Requests information Elaborates (self, other)
”Override”				
One-sided break				
Passive person				Consensual
Outside work				
The totality of the system		Prediction of result	Evaluating against goal	
Real world			Reasoning/ Inferring	
Syntax discussion				Reflects (standards, understanding)
Variables, methods, etc.				
			Other mental processes	Uncodable Evaluates (own, other, task) Regulates action Repeats (self, other)

Table 19: Coding schema literature summary table

Sallyann Bryant	**Sallyann Bryant, Pablo Romero, and Benedict du Boulay**	**Peng Xu Lan Cao**	**Carol K.K. Chan**	**S. Freudenberg (née Bryant), P. Romero, B. du Boulay**
Double trouble: Mixing qualitative and quantitative methods in the study of extreme programming	*The Collaborative Nature of Pair Programming*	*Activity Patterns of Pair Programming*	*Peer collaboration and discourse patterns in learning from incompatible information*	*Talking the talk: Is intermediate-level conversation the key to the pair programming success story?*
	Configure environment Correspond with 3rd party Find/ check example			
		Critiques	Patching	
Suggest/ counter. Confirm/ agree.			Stonewalling Surface assimilation	
	Agree strategy/conventions Discuss the IDE	Leader's activities Summary of results		
Test Reminding	Comment code Test Build, compile, check in/out Refactor Write new code Debug			
Question. Explain:		Ask for opinions Explanatory activities		
				Real world or problem domain
	Comprehend			Syntax
				Detailed Blocks of the program.
Other			Sub-assimilation Direct assimilation Surface-constructive Implicit knowledge-building Explicit knowledge building	Vague, including metacognitive statements and questions about progress or understanding.

X4. Complete Coding Schema Comparison Table

Nordbø, Walle — Collaboration in Pair Programming

Category	Description	Example
Other work	Small tasks relating to the experiment, but with no relevance to cooperation. Picking up printouts, asking the experimenter about the rules of the experiment etc.	
Break	Both participants taking a break. Physically absent from the workspace. No work-related activities in this period	
Private discussion	Things outside of the experiment are discussed. Not a break. Used only when what is discussed has no relevance at all to the work. For example must the phrase "Do you have much experience with Java?" be coded as "planning of future work".	
Disagreement that remains unsolved	A disagreement that ends with silence or inconclusive answer, or the actual statement: "let's move on" or similar, when it is clear that there is not an agreement.	
Disagreement solved	It is a disagreement from the beginning, but they agree after discussing it.	
Suggestion ignored	A suggestion from one is ignored by the other in some way.	
Suggestion accepted	A suggestion received a positive feedback.	
Planning of future work	Long-term discussion about long-term plans for the solution. Preferably with a 50-50 contribution rate to the discussion by the two participants, but a ratio as uneven as 80-20 is acceptable.	
Programming, duo	Both participants collaborate closely about code extension and are continuously discussion what should be done next and what is done. Actual programming (keyboard use directed towards the code) must be happening.	
Programming, solo	One person uses the keyboard and edits the code. At least one participant is silent. A possibility is that both are silent, and all one can hear is an occasional "mm-hmm". It could also be that one person comments what is being done, while the other is silent or ignores what is being said.	
Question answered	A question is answered, and the reply is understood/appreciated by the question asker.	
Question ignored	One person asks a question, but it is not answered, or answered so poorly that is does not benefit the questioner.	
"Override"	One person forces his opinion through.	
One-sided break	One participant takes a break, while the other continues the work, either by programming a bit alone or studying the problem (reading the code, the task description) or does other assignment related small tasks.	
Passive person	One of the two is not involved in the development/discussion, due to lack of knowledge, energy or other reasons	
Outside work	One of the people does private tasks during the pair programming session.	
The totality of the system	Connections between components; methods and classes. Discussions about a high-level "snapshot" of the program.	"What happens if I do this?"
Real world	Link between the real world and the task.	"OK. I'm going to buy coffee. Then I'll usually put on the money first, and then select the type of
Syntax discussion		"No, you have to use long there, not int.", "I spot a typo over here".
Variables, methods, etc.	The most abstract of the low level discussions. Here, the dialog is at method level, but still code focused and does not require a good mental mode.	"I guess it's getSugar we have to use here".

Gary M. Olson, Judith S. Olson, Mark R. Carter, and Marianne Storrøsten — Small Group Design Meetings: An Analysis of Collaboration

Category	Description	Example
Project Management	Statements having to do with activity not directly related to the content of the design, in which people are assigned to perform certain activities, decide when to meet again, report on the activity (free of design content) from previous times and	
Other	Time not categorizable in any of the previous categories. For example, in one meeting they discussed how their coordination procedures having to do with documentation were insufficient in the	
Goal	Statement of the purpose of the group's meeting and some of the constraints to work under, such as time to finish or motivating statements about how important this is.	
Digression	Members joking, discussion of side topics, or interruptions having to do with things outside the content of the meeting. When the person running the video camera speaks or changes tape, it is considered a digression.	"How to get the computer to make dotted lines" "why the plant was moved over by the window", "it is beginning to snow, we
Alternative	Solutions or proposals about aspects of the designed object. These are typically either features to offer the user or ways to implement the features decided on so far. This includes the elaboration of the idea, description not	
Criterion	The reasons, arguments, or opinions that evaluate an alternative solution or proposal. Occasionally these appear in the form of analogous systems, with the implication that if it worked in this other system, it will be good for this	
Issue	The major questions, problems, or aspects of the designed object itself that need to be addressed. This includes the elaboration of the idea, description not in answer to a group member's question. Occasionally, the issues are not stated	They typically focus on the major topics of "Shall we offer this capability to the user?" and "How can we implement
Meeting Management	Statements having to do with orchestrating the meeting time's activity, indicating that the group members are to brainstorm, decide (and vote), hold off on discussions, and SO on.	
Summary	Reviews of the state of the design or implementation to date, restating issues, alternatives, and criteria. It is a summary if it is a simple list-like restatement. If it is ordered by steps, it is a walkthrough, defined later.	
Walkthrough	A gathering of the design so far or the sequence of steps the user will engage in when using the design so far, used to either review or clarify a situation. It usually follows the user's task or the flow of data or messages inside a system	
Clarification	Questions and answers that someone either asked or seemed to misunderstand. This includes repetitions for clarification, associations, and explanations. Clarifications serve to clear up misunderstandings	

Takeshi Okada & Herbert A. Simon — Collaborative Discovery Scientific Domain

Category	Description	Example
(Disagreement) to a hypothesis.	When subjects expressed disagreement to the hypothesis or mentioned on alternative hypothesis in a unit. When subjects expressed agreement to	I don't think so.
Suspending conclusion.	When subjects mentioned suspending conclusion in o unit.	Let's not conclude now. We need more experiments.
Argument about justification.	If subjects argued against their justification, the unit is coded as having argument about justification.	
Agreement to the hypothesis.	When subjects expressed agreement to the hypothesis in a unit.	Yeah, right.
Hypothesis.	Hypothesis is a statement about effects of variables. When subjects mentioned a hypothesis in a unit, the unit is coded as having hypothesis.	The I gene chemically inhibits enzyme production.
Alternative hypothesis	When subjects mentioned two different hypotheses about a variable in a unit.	
Extension of the hypothesis.	When subjects added a new element to the hypothesis in a unit.	(original hypothesis: The I is chemical.) And, the 0 is physical.
Justification through experimental results.	When a hypothesis is accompanied by summary of results to justify in a unit. Justification with several results (next category) is included in this category.	The I is an inhibitor, since when the I is missing it produced a lot.
Justification using several experimental results.	When a hypothesis is accompanied by summary of results of more than one experiment in a unit.	The I is an inhibitor, since when the I is missing it produced D lot, while when I is there product was normal.
Plan for new experiments to test hypotheses.	When subjects planned to conduct an experiment to test o hypothesis in a unit.	Let's test if the I is really an inhibitor.
Summory of data	Description of data connected with condition. When subjects mentioned a summary of data in a unit.	When the I is missing, they produce 876.
Description of results	Description of result which is not connected with its condition. When subjects mentioned o description of result in a unit.	It's producing a lot. It produced before the lactose arrived at the chromosome.
Testability.	If subjects talked about whether a hypothesis can be tested or how it can be tested in a unit.	How can we test which of those two hypotheses is right?
Requests for explanation.	When subjects mentioned questions or metacognitive statement which request or imply people to generate further explanations in o unit.	I don't understand what this result means. How do you interpret these data? 50, what con we conclude now?
Prediction of result	When subjects predicted the results of the next experiment in a unit.	It must produce 876.

Kai H. Lim, Lawrence M. Ward, Izak Benbasat — An Empirical Study of Computer System Learning: Comparison of Co-Discovery and Self-Discovery Methods

Category	Description	Example
Formulating strategy/action	When the subject was developing a strategy or action plan to accomplish the goal:	1. We need to put the key to something else... maybe combining the key and the folder. 2. I think we better make a copy of
Describing physical action	When the subject was simply stating his/her physical movement	1. I am clicking on the button next to "Helen". 2. I am dragging the pen icon to the memo pad icon.
Reading screen output	When the subject was simply reading out the insructions or the output messages appeared on the computer screen.	1. Retrieve incoming memo. 2. This is not a required action.
Seeking Understanding Clarifying	When subjects attempt to seek understanding what needs to be done, or to clarify ambiguous points.	1. Am I gouing to be out of town in the 14th or the 15th? 2. I forget what this icon is for.
Interpreting output	When a subject's statement was related to understanding the feedback appeared on the computer screen as a result of the subject's action.	1. Um, what does that mean. 2. O.K., I see
Evaluating against goal	When a subject tried to compare the screen output against his/her intention to see if the goal had been achieved.	1. Good, I got it!. 2. Oh, this is not what I want.
Reasoning/ Inferring	When the subject made an inference, such as using an analogy based on his/her experiences, or expalining a logical and sensible plan/statement	1. Pencil is for writing something or drawing picture. 2. It can't be this icon because we tried that earlier and it was for mailing.
Other mental processes	If it did not fit into any one of the above categories, and the subject was clearly engaged in thinking about the task at the time the statement was made. Usually a general	1. I am totally confused. 2. I think I ha made a mistake along the line.

Kathleen Hogan, Bonnie K. Nastasi, Michael Pressley — Discourse Patterns and Collaborative Scientific Reasoning in Peer and Teacher-Guided Discussions

Category	Description	Example
Digressions		
Presents query		
Reacts (agrees, neutral)		
Reacts (disagrees)		
Presents (idea, partial idea, information, summary)		
Responsive	Both questions and responses of at least two speakers contributed substantive statements to the discussion. Although the roles of the questioner and responder	Student 1: The hotter a substance gets, the lighter it is. You know that, it has to, ya. Doesn't it go into the gas?
Elaborative	All speakers contributed substantive statements to the discussion, as in the responsive sequences. However, the speakers in elaborative sequences made multiple	Student 1: So in other words they look the same. Is that a gas, a liquid heated? Student 2: That's what I was talking
Requests information		
Elaborates (self, other)		
Consensual	When only one speaker contributed substantive statements to the discussion. Another speaker responded to the initiating speaker by (a) simply agreeing with the statement, (b)	Student 1: Alright, when a liquid is heated it turns into a gas, and then when a liquid is like frozen it gets cold and stuff, it
Reflects (standards, understanding)		
Uncodable		
Evaluates (own, other, task)		
Regulates action		
Repeats (self, other)		

Sallyann Bryant — Double trouble: Mixing qualitative and quantitative methods in the study of extreme programming

Category	Description
	Look up information. Either on-line or manually.
	Change driver.
	Rest
Suggest/ counter.	Suggesting a theory or approac or suggesting an alternative to an existing suggestion or explanation.
Confirm/ agree.	Confirm or agree with a previous suggestion or explaination
Test	
Reminding	Reminding other partner to do something, e.g Dont forget to fill in that stub function or to consider something e.g Dont forget that we or to consider
Question.	Direct Question e.g. What arguments does this function take? Or indirect question e.g. Could you give me a hand with this?
Explain:	Explaination of an approach or piece of existing code
Other	

Sallyann Bryant, Pablo Romero, and Benedict du Boulay — The Collaborative Nature of Pair Programming

Category	Example
Configure environment	Setting up paths, directories, loading software etc.
Correspond with 3rd party	Extra-pair communication: person to person, telephone or email
Find/ check example	Looking at examples in books, existing code or on-line
Agree strategy/conventions	Including approach to take, coding standards and naming conventions
Discuss the IDE	Talking about the development environment
Comment code	Writing or modifying comments in the code
Test	Writing, running and assessing the success of tests
Build, compile, check in/out	Compiling and building on own or integration machine
Refactor	Re-organising the code
Write new code	Creating completely new code to complete the assigned task
Debug	Diagnosing, hypothesizing and fixing bugs
Comprehend	Understanding the problem or existing code

Peng Xu Lan Cao — Activity Patterns of Pair Programming

Category	Example
Critiques	Modify solutions. Raise new problems. Explanations.
Leader's activities	Set up goal/tasks.Summarize current status.Formulating Strategy/Action.Reconcile differences.
Summary of results	Goal /task refinement. Mental model extension.
Ask for opinions	Ask for opinions
Explanatory activities	Request for explanation-Provide answer with explanation or justification-

Carol K.K. Chan — Peer collaboration and discourse patterns in learning from incompatible information

Category	Description	Example
Patching	Differences are compromised through using context-dependent examples.	
Stonewalling	Differences are rejected to minimize belief change	
Surface assimilation	Differences are ignored or conflated.	
Sub-assimilation	New information is reacted to at an associative level	I think the smaller you are, the faster you can go, like your strides are smaller . . . Giraffes have long legs and they have take bigger
Direct assimilation	New information is assimilated as if it were something already known.	Well, [the statement] is basically the same as what I thought. I know animals need to change with the environment
Surface-constructive	New information is comprehended, but its implications for one's beliefs are not considered.	I guess long legs helps reproduction because they run faster, they survive, and leave more offspring. It is not the deer's choice
Implicit knowledge-building	New information is treated as something problematic that needs explaining.	This card seems out of place from the others. I'm trying to piece things together into one whole, to find a connection. Right now,
Explicit knowledge building	New information is accumulated for constructing a coherent domain undertanding.	This card seems out of place from the others. I'm trying to piece things together into one whole, to find a connection. Right now,

S. Freudenberg (née Bryant), P. Romero, B. du Boulay — Talking the talk: Is intermediate-level conversation the key to the pair programming success story?

Category	Description	Example
		Savings account.
Real world or problem domain	The statement bridges or jumps between the real world or problem domain and the programming domain. This may be	So we need to add a test condition here, to see if the bank account is valid for this kind of transaction.
Syntax	Spelling or grammar of the program. Spelling is indicated in the transcriptions by single letter capitals. NOT semantics.	S P E L L I N G, dot, F9, 7.
Detailed	Refers to the operations and variables in the program. A method, attribute or object which may or may not be referred to by name.and Including tests and abstract coding concepts.	This condition, that return value, the list, the counter, what **this** returns or gives, getCustomer.
Blocks of the program.	Also strategy relating to the program and its structure. General naming	That loop, truncation, the error handling, Oracle, this issue. this part of the program, mock, Mosaic.
Vague, including metacognitive statements and questions about progress or understanding.	References to a place on the screen. References to the development environment and/or navigating it's menu structure.	Oh, yeah, I see, that bit at the the top.

Table 20: Complete coding schema literature summary table

X5. Calculation for Research Proposition P_2

Results from the regular analysis described in Section 13.2, including several insignificant findings, are summed up in Tables 21 and 22 below. Both of these tables are divided into two Tables a and b, below. Tables 21a and b contains the results indicating more dialog for different personality pairs, and Tables 22a and b lists indications that pairs of different personality discusses less. In both *a* tables are time used on the categories the measure, while in tables *b*; the number of times the category was used is analyzed. Non-significant results, based on the decision to use 0.05 as the confidence level limit, are listed in gray. A negative influence means that there is less of the second if there is more of the first. For example in the case of the fourth row of Table 21a, not counting the headline, a high difference in agreeableness indicates a reduced (not increased, because the row is marked as reverse) use of the category resolved.

R.1.2.1. Findings in favor of P_2

Difference in…	Influences time used on	Neg-ative?	Log Worth	P-value
Extraversion	Off-task		3.94	0.00
Openness to exp.	Responsive		2.10	0.01
Extraversion	Cross-purpose		1.51	0.03
Agreeableness	Resolved	√	1.28	0.05
Emotional stab.	Consensual	√	1.11	0.08
Extraversion	Unresolved		0.92	0.12
Openness to exp.	Flow	√	0.89	0.13
Openness to exp.	Disruption		0.89	0.13

Table 21a: Evidence of variable personality leading to more communication time

Difference in…	Influences occurrence of	Neg-ative?	Log Worth	P-value
Extraversion	Cross-purpose		2.00	0.01
Emotional stab.	Responsive		1.60	0.03
Agreeableness	Resolved	√	1.47	0.03
Extraversion	Nonresponsive	√	1.37	0.04
Extraversion	Unresolved		1.16	0.07
Conscientiousness	Consensual	√	1.11	0.08
Openness to exp.	Resolved	√	1.08	0.08
Openness to exp.	Flow	√	1.00	0.10
Openness to exp.	Disruption		1.00	0.10

Table 21b: Evidence of variable personality leading to more communication

R.1.2.1. Findings opposing P_2

Difference in…	Influences time used on	Neg-ative?	Log Worth	P-value
Agreeableness	Silence		1.35	0.04
Agreeableness	Stonewalling		1.27	0.05
Agreeableness	Elaborative	√	1.22	0.06
Conscientiousness	Prog. Silently		0.57	0.27
Emotional stab.	Nonresponsive		0.41	0.39

Table 22a: Evidence of variable personality leading to less communication time

Difference in…	Influences occurrence of	Neg-ative?	Log Worth	P-value
Openness to exp.	Elaborative	√	0.93	0.12
Agreeableness	Stonewalling		0.50	0.32

Table 22b: Evidence of variable personality leading to less communication

X6. Definition Based Hypothesis Suggestions

These definitions were made simply by making assumptions about the correlation between the definitions of the five personality traits and the collaboration categories. They are not based on any literature other than descriptions of the personality traits and the category definition list. Thus, they all receive low scores as sub hypotheses. Most proved to be wrong, but not all. Since these definition based sub hypotheses will not be discussed further, those marked with an asterisk are those which were significantly confirmed in the analysis. Those without an asterisk, the great majority of these sub-hypotheses, were not.

Each collaboration category is listed, and the personality traits that are likely to influence them are listed, one per line, with a score in front of them. A (-) denotes a negative correlation.

Not all categories from the schema are listed. Those not listed were viewed as most likely not affected by personality.

X6.1. Task Focus (TF)

X6.1.1. Off Task (Z)

The category is not really relevant for collaboration, but personality might influence it nonetheless. To make small-talk is elements of being extrovert, as stated in the literature and probably also agreeable, since it displays an interest for the other person. Conscientious people might se it as a waste of time, and people of high neuroticism might be uncomfortable with it.
3. Extraversion mean
3. Agreeableness mean

(-) 4. Neuroticism mean
(-) 4. Conscientiousness mean

X6.1.2. Task Description (D), Comprehension (C)

This category is about understanding the problem thoroughly and preparing to solve it correctly. It is feasible that conscientious people will feel that this is more important.
3. Conscientiousness mean

X6.1.3. Programming (P)

This category is mostly about ideas for what to do. To be full of ideas is one of the elements of openness.
3. Openness mean

X6.1.4. Programming Silently (PS), Silence (X)

Silent programming, compared to aloud programming, is a bit exclusive towards the other person. This would therefore perhaps be used more often by introverts and the not so agreeable. For silence, the same might apply.
(-) 2. Extraversion mean
(-) 3. Agreeableness mean

X6.2. Begin Characteristics (BC)

X6.2.1. Question (q)

The conscientious like structure and order, and will probably be interested in information just as much as suggestions for solution. Therefore, they might ask more passive questions.
3. Conscientiousness mean

X6.2.2. Assertion (a)

Suggestions and assertions imply that the person both actually has in idea, as well as the willingness to share it. People high on Openness or Extraversion might use this category more.
3. Openness mean
3. Extraversion mean

X6.2.3. Suggestion (s),

Same as for assertion, but for suggestion, the more diplomatic way of presenting it might be something that highly agreeable people would prefer.
3. Agreeableness mean

X6.2.4. Imperative (i)

The order-like tendency of imperatives might suggest that agreeable people would use it more, but like assertion, also openness and extraversion could influence it.
3. Openness mean *
3. Extraversion mean
(-) 3. Agreeableness mean

X6.3. Interaction Pattern (IP)

X6.3.1. Consensual (c)

Consensual answers are both those where it is an expressed agreement, as well as where the response is less enthusiastic, but there is at least some response. It will be hard to say anything about this category, but people of slightly low openness or extraversion might use it more.
(-) 4. Openness mean
(-) 4. Extraversion mean

X6.3.2. Stonewalling (s)

Probably the most common users of this category have a slightly low score on the trait agreeableness.
(-) 3. Agreeableness mean

X6.3.3. Cross Purpose (x)

It is likely that the people somewhat similar in personality and who have high values of certain traits and not so high on others, will *cross-purpose* discuss more often.
3. Extraversion mean
(-) 3. Extraversion diff
(-) 3. Agreeableness mean
(-) 3. Agreeableness diff
(-) 3. Openness diff

X6.3.4. Elaborative (e)

Like in the cross-purpose, the continuing discussion of an “e” might be preferred by people with somewhat similar personalities.
3. Extraversion mean
(-) 3. Extraversion diff
(-) 3. Agreeableness diff
(-) 3. Openness mean
(-) 3. Openness diff

X6.3.5. Nonresponsive (n)

Nonresponders might use this collaboration-less response since they are introverts and/or non-agreeable or they might lack understanding of what id discussed.
(-) 3. Agreeableness mean
(-) 3. Extraversion mean * (when measuring % of clips, not when measuring % of time)
(-) 3. Openness mean

X6.4. End Characteristics (EC)

X6.4.1. Disruption (d)

Similar to the cross-purposers, the disrupters might be very extrovert and not so agreeable, but the category will probably be found most frequently among those with a high openness score, since they have many ideas.
3. Extraversion mean * (when measuring % of clips, not when measuring % of time)
(-) 3. Agreeableness mean
(-) 3. Openness mean

X6.5. Result (Re)

X6.5.1. Unresolved (u)

Unresolved will perhaps happen very slightly more often when the programmers are very unlike each other, or when they have a low conscientiousness, openness or extraversion.
(-) 4. Conscientiousness mean
(-) 4. Openness mean
(-) 4. Extraversion mean

X6.6. Cognitive Level (CL)

X6.6.1. Program Model (P)

Conscientiousness people might be paying more attention to the program model, since they like details and order.
3. Conscientiousness mean

X6.6.2. Situation Model (S), Domain Model (D)

Openness might influence these, since it has to do with big ideas and abstraction.
3. Openness mean

X6.6.3. Metacognitive (M)

Metacognitive statements are probably made more often by extraverted people
2. Extraversion mean