

The Impact of Individual Differences on Pair Programming

Maija Kangas

Abstract— Pair programming is defined as two programmers working side by side at one computer, keyboard and mouse to produce an artifact. Although the success of the method obviously depends on the individuals practicing it, no covering survey has been made to clarify the impact of individual differences on it. This literature study explores psychological theories involved with group work to gain deeper understanding of the social aspect of pair programming. Hands on experiences concerning the compatibility of different pairs are summarized and some guidelines provided to take individual differences into account in pairing.

Based on the survey, the general recommendation is to pair together programmers with similar skill levels, personalities and self-esteem. Pair programming can be used as a way to promote team cohesiveness and thereby increase general productivity.

Index Terms— Group Process, Human Factor, Individual Differences, Pair Programming

1. INTRODUCTION

1.1 Background

Pair programming (PP) is a celebrated programming method. Williams & Kessler (2003) define it as two programmers working side by side at one computer, keyboard and mouse to produce an artifact. Programmers assume either the role of driver or navigator. Driver writes the code and the navigator searches for defects and thinks of alternative strategies and implementations. Roles can be switched at any time.

Agile software process introduced by Kent Beck and known as XP even recommends all code to be produced with pair programming. But is the method really suited for every individual and every pair of these individuals? What kind of programmers make a good pair? Studies on the limitations that different human factors pose on this method are scattered or cover only a very narrow viewpoint. It is hard to find any research papers, that would analyze who should be applying pair programming and with whom.

Although agile methods claim to put people before processes, at least XP doesn't seem to put individuals before methods. Pair programming as a method is

recommended for everyone in every situation, and the impact of individual differences on its use is only rarely considered in the same context. After all, pair programming is above all interaction between individuals – their characteristics do matter in making the method work.

The literature survey will summarize the empirically found impacts of individual differences on the success of pair programming. In addition, I will discuss applicability of psychological studies on this topic, especially considering the fields of collaborative decision-making and general teamwork. Group processes are discussed here, so that interaction in pairs and the way pairing affects the whole team can be better understood. The main consideration dwells on how individual characteristics and different combinations of them can support or resist these processes. There are two groups to consider when talking about pair programming: the pair itself and the development team in which pairing is exercised.

1.2 Research Problem

Research problem is established by the following research question and its subquestions:

- How do individual differences among programmers affect the successfulness of using pair programming?
- What are the characteristics desirable for a pair programmer?
- What are the characteristics undesirable for a pair programmer?

The terms *individual differences* and *successfulness* are defined according to the literature. The concept of individual differences will somewhat cover programmer's skill level, personality type, self-esteem, size of ego, gender, culture and age. Successfulness will cover efficiency and enjoyment experienced by programmers.

1.3 Objectives

The objectives of the study, in order of importance, are as follows:

- To gather information concerning the impact of individual differences on the profitability of pair programming through a literature survey, and to summarize the results.
- To explore the psychological studies made on related matters (pair/team work, collaborative problem solving) through a literature survey, and discuss their relationship to pair programming.
- To analyze the gathered information and propose some guidelines on how to take individual differences into account when applying or considering to apply pair programming.
- To introduce some topics to be further studied in the light of psychological theories.

1.4 Scope

Due to limited literature available on the subject, the scope of individual differences is not very wide nor deep. Only a few aspects are covered, and those are defined according to available literature. Also the success of pair programming is considered in the light of only a couple of success factors. The comparison of pair performance to individual performances is constrained, as the focus is on comparison among different combination of pairs.

The following questions, among many others, will not be answered, or they will only be implied to. Reasoning for leaving these out of the scope is provided for each question:

- In which circumstances should pair programming be used, meaning e.g. project type or phase? This question is too wide to be answered in this research, and should cover a lot of other aspects than individual differences.
- How does role differentiation affect the success of pair programming? The method itself includes the idea of switching roles regularly, so the question is a bit out of range.

1.5 Research Methods

Due to time limitations literature study is the only method used. This will limit the approach quite a bit, as the subject has not been widely studied. The narrow viewpoints presented in literature exclusively on pair programming will be complemented by psychological studies on related matters, like collaborative problem solving. The applicability of these theories on a special case like pair programming will be pondered.

2. PSYCHOLOGY OF GROUPS

2.1 Programmer as an Individual

Couger & Zawacki (1980) found that programmers have higher needs for personal growth and personal development than those in any other job category measured. However, they have lower needs for social interaction.

Curtis (1984) suggested, that the difference between an expert and novice programmer is that experts are better at encoding new information than novices. The more experienced programmers get, the more similar knowledge structures with each other they develop on a certain issue. However, expertise is specific to different knowledge domain, so a programmer can excel in one domain but be a novice in another.

2.2 Why Work in Groups at All?

Fisher & Ellis (1990) suggest, that groups can outdo individuals as the complexity of task increases. Groups have a larger pool of information and talent and can therefore resolve problems that individuals can't handle. Individuals also tend to oversimplify problems that are too complex for them. Even experts can become overconfident, misinterpret inconsistent evidence and make faulty conclusions. Groups specially outdo individuals in decisions that require judgment. When there are several solutions to an ambiguous problem, the decision requires more reasoning than expertise.

2.3 Can a Pair be considered a Group?

Shaw (1976) defines a group as "two or more persons who are interacting with one another in such a manner that each person influences and is influenced by each other person". Thus a pair, which in psychology is often called a dyad, is also a group. A dyad is just a simpler social system than a group of three or more, as there is only one communication channel and only two players. Mortenson (1972) indicates that a dyad is "a highly unstable, tension-producing form of interchange. It is unstable because everything depends on the continuous reactions of only one person – either A or B". In other words, both persons in a dyad have the power to cut all communication of the group at any time, which would mean a catastrophe for the group's functionality.

2.4 The Group Process

The central element of a group is the interaction between its members. Fisher & Ellis (1990) propose, that group process means individuals and their groups are continually changing and modifying themselves – change is the group's key characteristic and the group must be examined

as an entity that has developed over time.

Fisher & Ellis (1990) present that group process has two dimensions: task and social. Task dimension refers to the relationship between group members and their work – what they do and how they do it. Social dimension covers the relationships of group members with each other – how they feel about each other and the membership in the group.

Group dimensions can't be separated, but both exist in a group all the time. The outputs of the dimensions are productivity (task) and cohesiveness (social). Cohesiveness is covered more thoroughly later on: see chapter 2.5. The main rule is that productivity and cohesiveness have a two-way positive correlation – as the other increases, the other follows. However, there is one exception: highly cohesive groups are likely to have low or moderate productivity. One possible reason is that the purpose of the group turns into having a good time and the original purpose is left secondary. Groupthink is another threat: see chapter 2.6.

2.5 Cohesiveness

Fisher & Ellis (1990) define cohesiveness as 'the ability of group members to get along, the feeling of loyalty, pride, and commitment of members toward the group'. As cohesiveness increases, group members become more committed to the group's goal.

Cohesiveness can be promoted by increasing interpersonal attraction, member satisfaction and group identification. Group members with similar attitudes are more likely to be attracted to each other. In order to promote cohesiveness, it is best to compose groups of similar types of people whenever possible. Interpersonal attraction can also be supported by increasing the frequency of interaction and appropriate self-disclosure: see chapter 2.10.2. Member satisfaction is increased with e.g. participation in discussions and receiving feedback. A common "enemy", like a competing group, might produce group identification i.e. make its members feel like a part of the group.

2.6 Groupthink – the Dark Side of Group Decision Making

Groupthink is a phenomenon, where group avoids conflicts at the expense of evaluating ideas (Fisher & Ellis, 1990, p. 218) and thus results in a poor conclusion. Helkama et al. (2001) suggest that great coherence within the group, isolation from environment and external pressure expose groups to groupthink phenomenon. Fisher & Ellis (1990) present similar group norms that establish conditions for groupthink to occur:

- 1) Mindless cohesion – group is more interested in supporting cohesion than critically examining their decisions.
- 2) Pressuring nonconformists – oppressive atmosphere

towards dissenting opinions leads to self-censorship by group members.

- 3) Failing to reward critical thinking – opposition is not encouraged. These groups often have a strong leader who demands support, not conflict.
- 4) Tendency to justify what they have done – Group sticks to the idea, that they have made the best decisions. All information inconsistent with this idea is discounted.

Commitment to excellence creates a demanding group environment in which new and existing practices are appraised and challenged in a constructive way. This approach also reduces the risk of groupthink, as it generates a climate with good tolerance to diversity (West & Farr, 1990).

2.7 Roles and Personality Conflicts

A role can be defined as the group of norms that concern a status. Norm on the other hand is defined to be an expectation aimed at certain behavior.

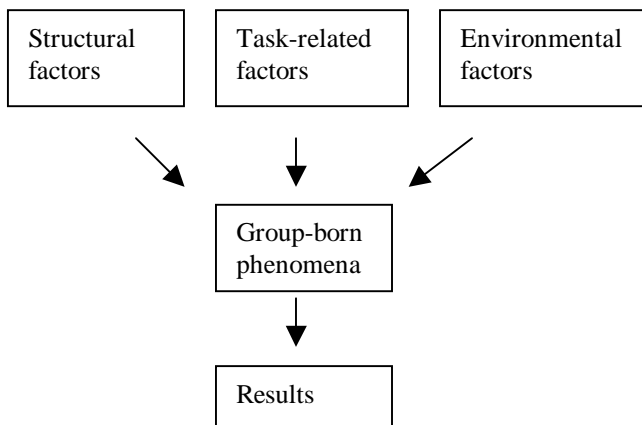
Problems arise when role and personality conflict. Role conflict can be seen as a planning problem of a social construction i.e. a managerial issue. Helkama et al. (2001) suggest two ways to reduce conflicts between personality and role beforehand. First, one can be trained for the role carefully. Secondly, if it is not easy to create the proficiency needed for the role, improve the methods for selecting a person for a role.

Role ambiguity is another source of unproductiveness and discomfort. Role holder needs adequate training to understand, how she is expected to behave in the role.

2.8 A Model of Group Efficiency Factors

Three initial factors affect the efficiency of group work (Picture 2.8.1): structural, task-related and environmental factors. Structural factors include e.g. group size, skills and personalities of group members and differences among the members. Difficulty and nature of the task in hand present task-related factors. In addition to physical environment, environmental factors consist e.g. of available technology and group's status in the organization (Helkama et al., 2001).

Initial factors form the basis for group-born phenomena, like cohesion, group motivation, norms and roles. The development processes of these are still quite blurred. Finally, group-born phenomena lead to results, including efficiency of performance, productivity and job satisfaction (Helkama et al., 2001).



Picture 2.8.1

2.9 Motivation Loss in Groups

According to Hertel et al. (2000), group work's profitability can be diminished by motivational loss. Motivation loss is affected by three phenomena, which all reduce individual motivation:

- 1) Social loafing – Individual believes, that her contribution cannot be detected or assessed.
- 2) Free riding – Individual believes, that her contribution does not matter in the group performance.
- 3) Sucker – Individual believes, that other group members are free riders and exploiting her. She doesn't want to be a 'sucker'. (Helkama et al., 2001).

In pair programming, motivation loss is mainly caused by the last two phenomena. Lesser-skilled programmers or those with low self-esteem might be prone to free riding. They feel their efforts wouldn't make a difference. On the other hand, experts might feel like suckers.

2.10 Interaction between the Individual and the Group

The relationship between an individual and her group is bi-directional. Individual members affect the performance of the group, and the group affects the self-concept of the individual. The relationship between the two can be seen as mutual development. Fisher & Ellis (1990) introduce four phenomena related to this relationship.

2.10.1 Social Comparison

Leon Festinger (1950, 1954) has argued that every human being has a need to evaluate herself. Often this need leads persons to evaluate their self-perceived abilities and qualities with those of other people, as there are very few if any nonsocial comparison methods. This

activity is called social comparison.

People tend to compare themselves with other people quite similar or with whom they are in close interpersonal contact. The notion of selectivity means, that person seeks to engage in social comparison with similar persons and tends to avoid comparison with those who are extremely dissimilar. If a person anyhow ends up in a social situation where comparison with very different people cannot be avoided, her self-concepts may change quite drastically.

2.10.2 Self-Disclosure

A person who reveals information about the private self is said to engage in self-disclosure. The phenomenon of reciprocity is closely knit to self-disclosure: as one person makes self-disclosures to another, the other typically responds with self-disclosure. Appropriate self-disclosure is quite delicate to timing, subject and the feedback it gets.

2.10.3 Interpersonal Trust

According to Fisher & Ellis (1990), there is trust in a relationship if the individuals believe each person acts in a mutually beneficial manner. Without a trusting relationship, groups are doomed to substandard performance. Developing trust is a process and thus it takes time.

Trust has two aspects: in addition to believing in the good will of the other party, individual has to believe both are capable of performing the task at hand. If you know the other party has never programmed and you two should develop professional-level software, there is probably little trust in the relationship no matter how devoted the beginner is.

2.10.4 Risk

Each commitment involves taking a risk – to avoid risk is to avoid commitment. Committing to group decision making exposes the individual to the risk of having her ideas rejected, which might lead to frustration, disappointment and ridicule or shame. Engaging in risk is prerequisite to an effective group process. Without agreeing to gamble there is no possibility of success for the group (Fisher & Ellis, 1990)

3. EXPERIENCES ON INDIVIDUAL DIFFERENCES AND PAIR PROGRAMMING

The discussion of individual differences inevitably leads to exploring different combinations of personal attributes. Most programmers with certain characteristics make an

efficient pair with someone – there are very few types of persons who gain nothing from pair programming. Most essential of those is a programmer with excess ego and/or “my way or the highway” attitude (Williams & Kessler, 2003; Jensen, 2003). Williams & Kessler (2003) also hint, that pair programming might not work for those who are too introverted, soft-spoken, lack confidence or feel threatened to expose weaknesses.

There are only few experiments that systematically study the impact of individual differences on pair programming. Most studies either select the pairings in random (Williams et al., 2002) or let programmers/students select preferred partners (McDowell et al., 2003). Several studies (e.g. Williams et al., 2000) suggest, that it takes some time for the programmers to adjust to pair programming. The benefits increase, as the pair gets familiar with each other and the method.

Nathaniel Talbott (quoted 2004) introduced his pairing experiences on his document *Conversant Pairing*. He found, that it is critical that both members of a pair maintain a high degree of interest and engagement with the problem at hand, so that enough of real feedback is provided. Pairs should be eager to explore alternatives and keep in mind that the goal is to produce the best code possible, not to be right. Humility is a good characteristic for a pair programmer. Things shouldn't be taken too serious to avoid unconstructive argument. Pairs shouldn't rush things, or they will end up simplifying the problem too much. Talbott finds that anti-social behavior is a big hindrance, varying from talking all the time to not wearing deodorant.

3.1 Skill Level

Jensen's (2003) experiment in a software organization suggests pairing programmers of the same experience and capability level is disadvantageous, the worst case being a pair of two “prima-donnas” who have strong egos. Best pair functionality is gained, when one member is slightly more capable than the other. A study by Katira et al. (2004) supports this view. According to them, student's perception of their partner's skill level has a major impact on their compatibility among all observed students (freshmen, undergraduate, graduate). Graduate programmers work well with partners of similar actual skill level, but there was not statistical significance among undergraduate and freshmen.

However, Gallis et al. (2003) found, that programmers considered that similar syntactical and technical knowledge were the most important requirements for efficiency. They discussed the human factor in pair programming based on several studies.

In an experiment by DeClue (2003) some students felt that if the other student was much more capable, she felt

hindered by the weaker student. Also Talbott finds working with someone at a significantly different level of expertise a hindrance to pairing, as it tends to reduce the amount of conversation. According to Talbott, the root of this phenomenon is pride. However, it can be seen as the individual's normal reaction, as she is forced to social comparison with an extremely different person.

3.1.1 Skill Levels According to Williams & Kessler (2003)

As long as mutual respect is present, *expert-expert* pairing can be extremely efficient and they can learn a lot from each other. Experts don't have to spend much time explaining their ideas to each other and they can concentrate on the main issues. Two experts are capable of solving even the most complex jobs fairly quickly. Combining two experts that have a very different viewpoint can be very beneficial. Biggest challenge is the size of egos.

Expert-average pairing is recommended to train the average person and still get the work done on an expert-level. However, Williams & Kessler suggest there are two types of average programmers: one is on her way to becoming an expert but lacks experience (wannabe-expert), the other has just remained average despite experience (just-average). Combining just-average and expert is just-frustrating, but expert-wannabe-expert pairing is an efficient way of training the average one.

Expert-novice pairing is a way to train the novice person and still get the work done on an expert-level. However, certain teacher-characteristics are required from the expert for this settlement to work out, like lots of patience, a willingness and ability to explain and compassion for the student. This pairing works only if the expert takes on the mentor role and accepts that her own productivity will be affected for the good of the long-term team performance. Student must not feel intimidated, or the pairing won't work and she might not learn at all. Williams & Kessler point out, that there are different kinds of novices: they might be excellent programmers who are not familiar with the precise technology used.

In both expert-average and expert-novice pairings, the expert needs some patience to slow down, answer the questions and explain what she is doing. The other person in the pair needs to be active and ask questions when she doesn't understand what is going on. However, the less-skilled party is not the only one to gain from the pairing, as the questions force the expert to examine the assumptions behind her solution – they might prove to be wrong. Experts can come to a better understanding of the techniques they are using, as they explain them out loud. However, this requires the expert is not too contemptuous of her partner, otherwise she won't be open to advice.

Novice-novice pairing can be an effective way to give them both valuable experience, but it is recommended, that there is an expert around to coach the pair when needed. Having a coach also improves the quality of the code a lot. Two novices will be much more efficient than one, because they have more courage to ask for help when they need it. However, novice-novice pair is not a productive way to produce code, as neither of them has enough experience to do an effective job.

3.2 Personality

In an experiment by Katira et al. (2004) freshmen worked better with partners with different Myers Briggs personality type. This result was not found with undergraduates. The Myers Briggs Type Indicator (MBTI) is used to classify people into one of the sixteen personality types based on four dimensions, one of which is introversion/extroversion. It was not specified more accurately, how the personality types in the experiment differed – they just weren't exactly the same.

3.2.1 Personality According to Williams & Kessler (2003)

As communication is the main point of pair programming, an *extrovert-extrovert* pair can discuss a large amount of ideas and openly question decisions, and finally come to the best solution. Feedback loops work great. However, extrovert pairs need to have a sense of self-discipline to keep the discussions limited and sensible, and remain productive. Otherwise, the pairing will not work out.

Extrovert-introvert pairing can work, if the programmers are around the same skill level, and both are willing to make some effort to recognize their personality. Extrovert needs to back off from talking all the time, and introvert must learn to open up when there are issues to discuss. Otherwise, the pairing won't work.

Introvert-introvert pair works best, when the programmers know each other and have developed other ways to communicate than speaking out loud. Pairing an introvert with another is also a good way to introduce one to pair programming. With an extrovert, the introvert would have to compete to get a word in and might give up. Introverts have a longer pair-jelling period, during which they might not communicate enough. Afterwards they will be quite productive, as they can be very intense on the task. However, introverts are very likely to resist pair programming and some will never be able to pair efficiently.

3.3 Self-esteem / Attitude Towards Programming

Katira et al. (2004) found, that student's self-esteem does not have any major influence to pair compatibility.

Although students were paired randomly, around 90% of them reported that they and their partners work compatibly.

A study by Thomas et al. (2003) suggests otherwise. First year students answered a questionnaire to place them on a 9-level scale between Code-a-Phobe and Code-Warrior, that reflected their attitude towards programming, a certain "programming self-esteem". For the first pairing experiment, the opposite attitudes were put together. In the second experiment, these same students got to pair with someone of their own attitude level. After the first assignment, overall 66% enjoyed the experience and 66% found it led to a better solution, but the figures for Code-Warriors were only 53% and 47%. Combinations of warriors and middles got the highest grades and phobe-warrior groups the lowest.

Table 3.3.1 Percentage of Students enjoying pair programming (Thomas et al., 2003).

	All (%)	Code-Warriors (%)
1 st Exercise	66	53
2 nd Exercise	64	58

Table 3.3.2 Percentage of Students thinking pair programming resulted in a better solution (Thomas et al., 2003).

	All (%)	Code-Warriors (%)
1 st Exercise	66	47
2 nd Exercise	65	67

After the second exercise, the overall enjoyment percent was 64% and 65% thought it led to a better solution. Only 58% of the warriors enjoyed the experience, but 67% of them thought the solution was better. Overall, the second exercise was preferred to the first one. Especially phobes liked pairing with similars more. Grades were essentially the same with all groups, although the exercise was easier too. Thomas et al. draw the conclusions, that student's self confidence and the enjoyment they get from pairing is inverse. It also seemed, that students produce their best work when paired with students of similar levels of confidence.

According to Williams & Kessler (2003), excess ego is a personal attitude problem. If you can get the individual to admit the problem, there might be hope, but most often this is not the case. People with too little ego, meaning low self-esteem, will not contribute much to pair programming. If they are paired with a confident person with good people-skills, and forced to drive, there is hope of improving. The reason for the low contribution is low commitment, which follows from fear of failing and being ridiculed.

3.4 Gender, Culture and Age

Pöyhönen (2001) studied knowledge workers in her thesis, and found that low level of innovation is significantly connected to age, sex and even highly job-

related heterogeneities like educational field. This speaks for the superiority of homogenous groups, as the metric is innovation. Interpersonal trust among team members had a strong positive correlation to innovation.

Williams & Kessler (2003) suggest, that gender is not an issue unless gender chauvinism exists. They apply the same to culture: as long as the pair can communicate both verbally and nonverbally and there is no cultural bigotry, culture doesn't matter – adapting to co-work just might take a little more time. Williams & Kessler (2003) see pair programming as an efficient way to eliminate cultural barriers and build trust within the team.

3.5 Patience and Trust

According to Williams & Kessler (2003), a so-called Professional Driver Problem exists, if the other partner hasn't got enough patience to keep her hands off the keyboard, doesn't trust the navigator or the navigator doesn't trust herself. If they can't get over it, it's better to not let them pair program.

4. GUIDELINES FOR USING PAIR PROGRAMMING

The ideal pair programmer could be a patient, devoted, humble expert with good social skills, a good sense of humor and an ability to give excellent feedback. The worst-case scenario could be an anti-social, smelly novice with a huge ego or an extremely low self-esteem. For most qualities, the same problems that come up with pair programming, are encountered already on the team level. They just might not be as evident or significant in a bigger group, as one person's contribution to the group functionality is relatively smaller and her interaction with the rest of the group can be adjusted to an appropriate level or restricted to certain communication channels.

Generally speaking, most pairings work out fine after some adaptation, but some personalities and combinations

just don't match. As a golden rule, combine similar people to be effective and to optimize the enjoyment experienced by programmers. However, it must be bared at mind that no individual characteristic exists in a vacuum, and only few can be interpreted on a yes/no scale. It is not clear at all, how different characteristics affect each other inside one person. Above all, individuals consist of individual combinations of individual characteristics, and the outcome does not follow a mathematical model. A suggestive summary of the literature survey is presented in *Table 4.1*.

Here are some introductory guidelines for using pair programming:

- Two extroverts are a great match because of extensive feedback loops, but check on them once in a while. Their discussions get easily carried away.
- Pair introverts with someone they trust or be prepared for a long adjustment period.
- Introvert - extrovert pairs should pay special attention to even communication. There's a risk that introvert stays silent as the extrovert talks all the time.
- Two introverts make a very efficient an intense pair, although the start may be crummy.
- Combine people from around the same skill level for efficiency and enjoyment.
- If your goal is to train the less-skilled person, it requires patience from the trainer and humbleness from the trainee. Relatively small skill differences are preferable. Don't match experts with novices.
- Don't let two novices pair, unless you can provide an expert to tutor them when needed.
- People with very low self-esteem don't commit to the pair work and can become passive. However, a person with very good social skills could prove an effective pair.

Table 4.1 Suggestive summary of the impact of individual differences on pair programming

	Experience	Psychology	Jelling time	Enjoyment	Efficiency	Special Characteristics
<i>Personality:</i>						
Introvert - Introvert	+	+	Long	+/-	+	
Introvert - Extrovert	+/-	-	Long	-		
Extrovert - Extrovert	+	+			+/-	Prone to babble – keep watch
<i>Skill levels:</i>						
Expert - Expert	+	+			+	
Expert - Average	+/-	-			+/-	Requires mentor attitude
Expert - Novice	-	-			-	Requires mentor attitude
Novice - Novice	-	+/-			-	Expert coach required
<i>Self-esteem:</i>						
Low - Any	-	-	Long			
<i>Size of ego:</i>						
Big - Any	-			-		Not suitable for PP
<i>Others:</i>						
Different cultures	+/-		Long			
Different ages	(-)		Long			
Different gender	+/-		Long			

- Never let a person with excessive ego pair program – or otherwise closely interact with others
- Combine people with different cultures, ages and gender with reservation. Stay clear of any bigotry. After a longer adjustment time, these pairs should work fine.
- Support a democratic working environment and commit to excellence to avoid groupthink. Be alarmed, if the pair seems to agree on everything all the time.
- Train the programmers on pair programming practices and communication skills. This helps to avoid role ambiguity and conflicts between roles and personalities.

On the other hand, pair programming affects individuals and the whole team dynamics and not just vice versa. Pair programming can be used as a means to enhance individual abilities and harness them to serve the team. Some team-level problems caused by individual differences could actually be resolved with pair programming. Use pair programming to

- smooth down skill differences between programmers
- build trust in the team and thereby enhance innovativeness
- increase cohesiveness and thereby productivity
- encourage programmers with low self esteem

5. DISCUSSION

Literature survey always sets some limits to the research. The biggest problem with this one was the lack of systematic experiments that would concentrate solely on the impacts of individual differences on pair programming. Several studies discussed the effects of pair programming in general, and only briefly mentioned that some observations were made on the compatibility of certain types of pairs. One of the central resources of my study (Williams & Kessler, 2003) was actually mainly based on subjective, although experienced, views of the authors. In the light of other sources, the spirit of the book seems a bit optimistic.

Some studies reported results poorly, e.g. accurate combinations of personality types were not mentioned in the study by Katira et al. (2004). In a few studies, pairs were selected randomly or according to preferences, and the final combinations were not always reported. Some experiments selected only those people to try pair programming that expressed interest in the method. This might indicate those individuals that might cause the

biggest problems when paired are not included in the study, as they might not be that interested in pair programming in the first hand.

The survey by Pöyhönen (2003) discusses knowledge workers in general, and its applicability to programmers is debatable. On the other hand, it was one of the few studies which had been conducted in a working environment. Most experiments were done on students and their tenability in a professional environment might not be straightforward.

A more extensive study on psychological literature could have exposed more fitting theories and experiments on pair work particularly. As a pair is a very exceptional group form, the presented theories might have given a slightly distorted picture of its functionality.

Psychological theories and pair programming experiences go mainly hand in hand – most deviances seemed to rise from Williams & Kessler (2003). Both psychological models and practical experience support the fact, that forming a functional pair takes time. Psychology approaches the phenomenon as the absence of social structure that needs to be developed. On the other hand, pair needs to develop a sense of interpersonal trust and increase cohesiveness to be productive. If the pair is familiar with each other, this phase can be shortened. More long-term experiments would be needed to observe the effects of individual differences in a mature phase of group development.

Based on the studies, it can be speculated that programmers with low self-esteem and maybe even some introverts have a hard time taking risks and trusting people, which is the reason to an extended group formation time. This time is mainly unproductive and at the worst unsuccessful – studies showed that some people with low self-esteem never profit from pair programming.

Psychology provided several reasons for encountered conflicts between programmers with different skill levels. Theory of social comparison has the greatest explanatory power for why people enjoy the company of similar people. It explains many phenomena observed in practical pair programming experiments, e.g. why programmers with the same skill level, attitude and external qualities (culture and sex) work well together. According to Pöyhönen (2001), similarity also enhances innovation.

Programmer's higher need for personal growth and development is another source of conflict, as experts might feel hindered by less capable partners. Experts can also suffer from motivational loss if they feel they are the "sucker" in the pair. From the viewpoint of cognitive psychology, different level pairings work badly together because of their dissimilar knowledge structures. On the other hand, even experts make obvious mistakes due to the complexity of programming, and might therefore gain something from lesser-skilled partners. Great differences in capability level also hinder interpersonal trust, if the lesser

skilled can't be trusted to handle her task.

The idea of role conflicts rises the question, if it would be beneficial to regulate the switching of driver and navigator roles. In my opinion it seems likely, that people with low self-esteem could overcome their fears and get committed to the pair work, if they were drivers. As navigators, they, as well as introverts, might just stay quiet. On the other hand, it could be too traumatic to be put in an active role straight away.

The division of skill levels to novice – average – expert is quite vague. Based on the literature, it is not clear how an expert or her expertise domain is defined or identified.

Although individual differences play a certain role in group's functionality, they alone don't determine the results of the co-work. As Helkama et al. (2001) presented in a model of group efficiency factors, it's the group-born phenomena that determine the outcome in the end. Initial factors cover several other aspects than individual differences, and some of those factors, like organizational status and tools, can be changed to support the group process. Thus, the efficiency or enjoyment of any certain programming pair can't really be judged in the beginning. However, as a pair is more unstable than a bigger group, the impact of individual differences is increased.

Last but not least, when discussing the impact of individual differences on pair programming, it must be bared in mind that pair work also affects the individual. Most characteristics of a person are not stable, but develop along with the relationship of the pair. Actually, the title of this survey could as well be "The Impact of Pair Programming on Individual Differences".

6. CONCLUSION

The purpose of this literature study was to explore the relations between individual differences and successfulness of pair programming. Psychological theories involved with group work were explained to gain a deeper understanding of the social aspect of pair programming. Hands on experiences concerning the compatibility of different pairs were summarized and some guidelines provided to take individual differences into account in pairing. Based on the survey, the general recommendation is to pair together programmers with similar skill levels, personalities and self-esteem. Pair programming can be used as a way to promote team cohesiveness and thereby increase general productivity.

Further studies with systematic pairings and definition of individual differences are needed – both dependent and determining variables should be defined and measured accurately. There are no studies that explicitly explore the impact of individual differences on software quality, e.g. the relationship between different personality type combinations and bug rate. In addition, more studies in

industry are required, as most are conducted in a university environment. Experiments should also cover longer time periods to observe the effect of group development. Most present studies have lasted such a short time that the results cover only the pair jelling phase.

REFERENCES

- Brown, K., Klastorin, T., & Valluzzi, J. 1990, "Project Performance and the Liability of Group Harmony", *Engineering Management, IEEE Transactions on*, vol 37, is 2, pp. 117-125.
- Cheng, M., Luckett, P., & Schultz, A. 2003, "The Effects of Cognitive Style Diversity on Decision-Making Dyads: An Empirical Analysis in the Context of a Complex Task", *Behavioral Research in Accounting*, vol 15, pp. 39.
- Cockburn, A., & Williams, L. 2001, "The Costs and Benefits of Pair Programming", Addison-Wesley Longman Publishing Co., Inc.
- Curtis, B. 1984, "Fifteen Years of Psychology in Software Engineering: Individual Differences and Cognitive Science", *Proceedings of the 7th international conference on Software engineering*, IEEE Press.
- DeClue, T. 2003, "Pair Programming and Pair Trading: Effects on Learning and Motivation in a CS2 Course", *The Journal of Computing in Small Colleges*, vol 18, is 5 (May 2003), pp. 49-56.
- Fisher, B., & Ellis, D. 1990, "Small Group Decision Making", McGraw Hill Publishing Company, 3rd ed.
- Gallis, H., Arisholm, E., & Dybå, T. 2003, "An Initial Framework for Research on Pair Programming", *Proceedings of the 2003 international symposium on empirical software engineering*, IEEE Computer Society.
- Hazzan, O. 2003, "Cognitive and Social Aspects of Software Engineering: a Course Framework", *Annual Joint Conference Integrating Technology into Computer Science Education*, ACM Press.
- Helkama, K., Myllyniemi, R., & Liebkind K. 2001. "Johdatus sosiaalipsykologiaan", Oy Edita Ab, 4th ed.
- Hertel, G., Kerr, N.L., & Messé, L.A. 2000. "Revisiting the Köhler effect: Does diversity enhance group motivation and performance?" In S. Stumpf & A. Thomas (eds.), *Diversity and group effectiveness*. Lengerich, Germany: Pabst Science Publishers
- McDowell, C., Werner, L., Bullock, H., & Fernald, J. 2003, "The Impact of Pair Programming on Student Performance, Perception and Persistence", *Proceedings of the 25th international conference on Software engineering*, IEEE Computer Society.
- Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. 2003, "Improving the CS1 Experience with Pair Programming", *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, ACM Press.
- Jensen, R. 2003, "A Pair Programming Experience", *CrossTalk, The Journal of Defence Software Engineering*, March.
- Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., & Gehringer, E. 2004, "On Understanding Compatibility of Student Pair Programmers", *SIGCSE '04*. ACM Press.
- McDowell, C., Werner, L., Bullock, H., & Fernald, J. 2002, "The Effects of Pair Programming on Performance in an Introductory Programming Course", *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, ACM Press.
- Pöyhönen, A. 2001, "Correlates of Innovation in Knowledge Worker Teams"
- Talbott, N. "Conversant Pairing", <http://www.pairprogramming.com/conversantpairing.htm>, 8-3-2004. Note: no creation date available for this work.
- Williams, L., & Kessler, R. 2000, "All I Really Need to Know about Pair Programming I Learned in Kindergarten", *Communication of the ACM*, ACM Press.

- Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. 2000, "Strengthening the Case for Pair Programming", IEEE Software, vol. 17, no 4, pp. 19-25.
- Williams, L., & Upchurch R. 2001, "In Support of Student Pair Programming", Proceedings of the 32nd SIGCSE technical symposium on Computer science education, ACM Press.
- Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. 2002, "In Support of Pair Programming in the Introductory Computer Science Course" Computer Science Education.
- Williams, L., & Kessler, R. 2002, "Pair Programming Illuminated", Addison-Wesley Longman Publishing Co., Inc.
- Thomas, L., Ratcliffe, M., & Robertson, A. 2003, "Code Warriors and Code-a-Phobes: a Study in Attitude and Pair Programming", in Proceedings of the 34th SIGCSE technical symposium on Computer science education, ACM Press.
- West, M. & Farr, J. 1990, "Innovation and Creativity at Work", John Wiley & Sons