Evaluation of pair programming

Pim Erlandsson, pime@kth.se
Sam Khosravi, samkh@kth.se

Modern mjukvaruutveckling
IV1303

Abstract

This report will cover the topic of pair programming in regards to a machine learning and algorithm heavy topic. We seek to find the benefits of different types of pair programming methods and to compare these with each other. Our own experiences will be documented and compared to international publications. The report extensively covers the background of pair programming, the pair programming implementation in our 4 weeks of coding, our project, what research method we used and how we did it, and lastly our results and evaluation of the project work. The overall goal is to find out how effective pair programming is, and what type of specific pair programming method worked the best for us.

Keywords

- Flow based pair programming
- Timed pair programming
- Individual programming

SECTION I: Introduction

The common notion of programming has since long been to alone sit and write code tirelessly for hours. The work being made is certainly discussed together with co-workers, and the main goal of a project will still be discussed tirelessly between the teams on a daily basis. However, the line by line writing of the code is done by one person. In very rare cases a programmer might ask his peers for help, but in true programmer spirit, a person is supposed to tackle a task on his own.

However, this common notion has recently been challenged by advocates of pair programming. Pair programming is, as the name suggests, the act of the people sitting side by side and writing code together. The code is written on one computer by one person, while the other one is observing. The praxis is however not that one person writes and one watches, but the work is instead heavily monitored by both programmers. The two programmers work together, discussing ideas and sharing knowledge with each other. With different styles of pair programming comes different approaches.

The most common way of pair programming is to use the metaphor of a “driver” and “navigator” (Chong & Hurlbutt, 2007). The driver will be the one working on the implementation and actual line by line writing of the code, while the navigator sets up the work in a strategic manner and to keep a comprehensive logic throughout the program. Another method presented in the course IV1303 was to give each programmer a set amount of time, usually 20 minutes to code, and when the time was up the seats were switched. In addition to this, there are many other ways of pair programming. The methods used during this course was to the highest extent the more natural method, where both members worked on the exact same problem on the same logical level, and the division of code writing was partitioned based on natural flow. However, the timed approach to pair programming was also tried for a while, but was found to be ineffective and disregarded after a few tries.

The main problem of pair programming is that it is considered to be ineffective, as if two people work together writing code their productivity will be cut in two. In addition to this, we

have no prior experience with pair programming in a professional setting. The goal with this study is to evaluate the effectiveness of pair programming through various research and practitioner findings, but also through our own experiences throughout the project.

SECTION II: Background

Software development methods

Traditional development methods are used when it is clear what the end product is well understood by everyone. This traditional software development method requires pre-arranged phases and stages of the software lifecycle. Furthermore, each phase of the model contains detailed documentation and specific deliverables that have gone through a rigorous review process. (Mavuru, 2018) The method is a linear approach, meaning that all phases of development must be completed in sequential order (Morelos, 2018).

However, agile software development methods are focused on the idea of iterative development, meaning that requirements and solutions are evolved during the development where self-organizing cross-functional teams collaborate.(Cprime, 2022) This method requires more customer involvement since their input is needed during the stages where requirements are being gathered due to the fact that they need to provide detailed explanation of the requirements.

Unlike traditional development methods, agile development methods gives the users or customers opportunities to make modifications during the different phases of development due to the fact that the phases does not need to be pre-organized and must not go through a review process causing the development method to become flexible. This makes the agile development methods more customer friendly. Another difference between the methods is that documentation is a high priority in traditional development methods. The documentation is time consuming and expensive. (Mavuru, 2018)

Software development practices

Software development practices are a large variety of concepts, principles, methods and tools that must be considered during the planning and development of software (Alkadi, 2013).

These are list of the most known practises:

- Pair programing
- Test-first development
- Iterative development
- Retrospectives
- Informative workspaces
- User Stories
- Story points

- Estimation
- Planning
- Risk management
- Release planning
- Iterative acceptance testing
- Refactoring
- Communication

Out of the listed practices, the report will focus on pair programming.

Pair programming originates from Extreme Programming and is an Agile software development practice. This concept is the idea of two programmers teaming up on one computer working together to write and design code in order to make this process more efficient with the knowledge of two programmers combined. The implementation of pair programming can be done in many different ways. However, a common way of implementing this concept is calling the programmer at the keyboard the “driver” and the other programmer sitting next to the “driver” the “navigator”. The task of the navigator is to focus on the overall direction of the programming while the driver is the one actually implementing the code. The two programmers take turns coding, meaning that the programmers switch between being the driver and the navigator. This work requires the programmers to communicate a lot to help each other to face different problems that might occur during the coding. (Gillis, 2021) Pair-programming promoters argue that this method of development is beneficial when it is applied to develop new code and to preserve or to amplify already existing code. Benefits of pair programming are by these advocates of pair programming stated to be better teamwork, enhanced knowledge transfer, and improved learning. However, others argue that pair programming is the least beneficial of all Extreme Programming (XP) practices. It is furthermore stated by the same people, that beginners, paired together is similar to “the blind leading the blind” which is an undesired combination. (Hannay, Dybå, Arisholm & Sjøberg, 2009)

Presentation of our project

Our project consisted of creating a parking app that could read parking signs. The user should be able to use the app on Android and IOS. When the user is logged in, the app should provide the user with a “parking near me” feature and a “scan sign” feature. Selecting the parking near me feature would show the current position of the user and parking nearby. The “scan sign” feature would let the user take a photo of a sign and get to know whether the user can park there at that current time or not.

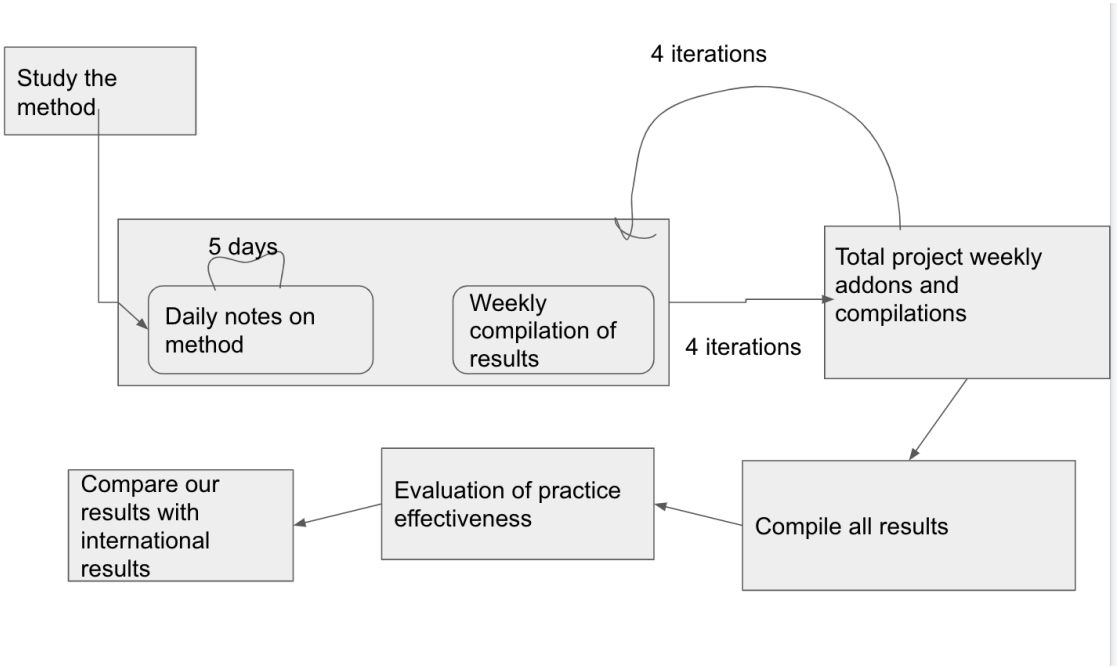
The project is done using Unity for the front end. This is where the app is created and the programming-language “C#” was used to develop the app. Unity is in essence a game development tool, but is also very convenient when wanting to integrate one programming language into both IOS and Android apps, instead of writing the android code in Java and the IOS code in Swift. When the user takes a photo in the app, the photo is sent to a server where AI is used to scan the sign. This is the backend of the program which is far more extensive than the frontend.

The Backend consists of an AI, which in turn consists of Detectron 2/d2go for image scanning, and OCR for text detection and interpretation. The models need extensive training to be functional, which would consist of different datasets of parking signs which were manually labeled. The labeling was made by the software program “LabelIMG”. Up to a thousand pictures were collected in the form of pictures of different signs. This data is used to train a good model for the AI in order to make it more accurate. The AI is trained to recognize different parking signs and is able to extract important information from it. In order to tell the user whether it can park or not, the extracted information is formatted in a string and then sent to an algorithm that is developed by our team from scratch.

The information is sent to the algorithm and is parsed and sent through a number of methods and pattern-matched to find out what specific sign is being processed. For more complicated signs where there is an abundance of information the same method is used, and the individual parts of the signs are sequentially gone through and pattern matched. The information given in total from the sign as a whole is then processed and a string is given out in the form of if you can park there or not, for how long and if there is a fee. The algorithm is written in python in order to make the integration with the AI easier since it is written in python as well. The result is afterwards sent to a local server which in turn makes the string visible in the app.

Furthermore, a database is developed where different locations of parking signs are stored. The database is developed using postgreSQL and Java, and is also connected to the server. The location of each picture taken is taken by the app, sent to the server, which in turn sends it to the database. The server and database communicate using 1’s and 0’s, where 1 means that the server wants to save information and 0 means that the server wants to extract information.

SECTION III: Research Method



The research method consisted of the above steps. The method of pair programming was to a start researched extensively, and many different aspects and types of pair programming were looked over. The choice of method was not set in stone and mainly two methods were chosen to be trialed. One being to pair-program dependent on flow and the other method was to pair-program with a timer which would decide when the physical coding was to be switched between each programmer. To get a good grasp of each method not only research into many different international publications were made, but also anecdotes from various famous publishers in the area. Lastly, our own experiences from earlier course work and professional experiences were taken into consideration, as both authors of this paper have now completed two years of the degree program in information and communication technology at KTH.

The next phase consisted of daily note taking. Oftentimes notes would be taken throughout the day whenever deemed necessary. The notes were written down on small post it papers. This was done everyday, and as such would be made with 5 iterations a week. At the end of each day or just spontaneously during the day the post-it notes were compiled into a paper with bullet points. No coding was done on each Friday, instead mock pair programming was done to further improve and get more data. The mock pair programming was done on very small projects that would imitate the real project. As the project consisted of many different algorithms and pattern matching, the smaller mock projects on Fridays would consist of writing algorithms with a time complexity less than O(n) and to include some type of pattern matching in them. However, this was only a small part of the slack time given on Fridays, as even half an hour would highly benefit the amount of data needed to get an accurate conclusion. Furthermore, each Friday a weekly compilation of results would be made and in turn discussed extensively. Each compilation would be saved in a Google Docs document. This is because each weekly compilation would be compared to earlier ones. This is in the above figure seen as the “Total project weekly addons and compilations” part, where the weekly compilation results would be composited in a new document.

At the end of the project course all results were compiled into a google drive document. The evaluation was done through discussion between the two group members, and each week was rated and ranked according to what methods were the most effective and how each week differed from another. The results were then compared to international scientific findings.

The following are the main criteria that were used for evaluating the practice of pair programming.

Correctness: This evaluation criteria refers to the amount of errors in regard to other parts of the code and overall errors. In this report, pair programming will be evaluated in regards to correctness to see if it improves overall software correctness. This criteria is included because the less errors a software program has, the more functional it is.

Amount of time required for code to be written: This evaluation criteria refers to how many qualitative lines of code were written and how fast, and in whole how fast different parts of

the software were written. All software is written in code, and the faster a qualitative line of code is written the faster the project is finished.

How fast problems were solved: This evaluation criteria is the most important one, as it will in majority be the deciding factor to if pair programming is better than programming alone. This as all problems are first solved outside of the software program either through dialogue or inner monologue. The faster a problem is solved the faster the programmer will be able to start implementing it in code.

Section IV: Project results

The results after 4 weeks were evaluated based on the evaluation criteria. The first evaluation criteria is correctness, which refers to the amount of errors present. The errors for each day and week are presented in table form instead of in running text, as no intricate explanation should be needed.

The resulting errors are the ones present at the end of the pair programming session when one member gives over the computer to the other one. As many sessions were had during the day, the errors were added together to give a number of total errors during the day. The following four tables are based on the first evaluation criteria which is Correctness. No data was used based on individual programming, as even though this method was trialed, we noticed a huge difference in productivity when pair programming instead of doing so individually. Therefore, more relevant data would come in the form of comparing different pair programming methods.

Table 1: illustrates results week 1

Results week 1	Flow	Time
Monday	Errors: 17	Errors: 14
Tuesday	Errors: 8	Errors: 37
Wednesday	Errors: 14	Errors: 21
Thursday	Errors: 16	Errors: 13
Total errors	55	85

The above table illustrates the results from week one. During the first day of the week, Monday, the total amount of errors amounted to 17 for Flow pair programming without any time limit, while the amount of errors amounted to 14 for pair programming based on a timer. For Tuesday, the amount of total errors amounted to 8, while timed pair programming had a total number of errors amounting to 37. For Wednesday, the total number of errors for flow pair programming amounted to 14, while the total number of errors for timed pair programming amounted to 21. For the last day of coding, Thursday, the amount of errors

amounted to 16 for flow pair programming, while timed pair programming had 13 errors. The amount of error for pair programming based on flow resulted in a total of 55 errors for week 1, while timed pair programming had a result of 85 errors for week 1.

Table 2: illustrates results week 2

Results week 2	**Flow**	**Time**
Monday	Errors: 8	Errors: 17
Tuesday	Errors: 9	Errors: 21
Wednesday	Errors: 16	Errors: 22
Thursday	Errors: 10	Errors: 11
Total errors	43	77

The above table illustrates the results from week two. During the first day of the week, Monday, the total amount of errors amounted to 8 for Flow pair programming without any time limit, while the amount of errors amounted to 17 for pair programming based on a timer. For Tuesday, the amount of total errors amounted to 9, while timed pair programming had a total number of errors amounting to 21. For Wednesday, the total number of errors for flow pair programming amounted to 16, while the total number of errors for timed pair programming amounted to 22. For the last day of coding, Thursday, the amount of errors amounted to 10 for flow pair programming, while timed pair programming had 11 errors. The amount of error for pair programming based on flow resulted in a total of 43 errors for week 2, while timed pair programming had a result of 77 errors for week 2.

Table 3: illustrates results week 3

Results week 3	Flow	Time
Monday	Errors: 11	Errors: 24
Tuesday	Errors: 7	Errors: 19
Wednesday	Errors: 10	Errors: 18
Thursday	Errors: 15	Errors: 33
Total errors	43	94

The above table illustrates the results from week three. During the first day of the week, Monday, the total amount of errors amounted to 11 for Flow pair programming without any time limit, while the amount of errors amounted to 24 for pair programming based on a timer. For Tuesday, the amount of total errors amounted to 7, while timed pair programming had a total number of errors amounting to 19. For Wednesday, the total number of errors for flow pair programming amounted to 10, while the total number of errors for timed pair programming amounted to 18. For the last day of coding, Thursday, the amount of errors

amounted to 15 for flow pair programming, while timed pair programming had 33 errors. The amount of error for pair programming based on flow resulted in a total of 43 errors for week 3, while timed pair programming had a result of 94 errors for week 3.

Table 4: illustrates results week 4

Results week 4	Flow	Time
Monday	Errors: 18	Errors: 49
Tuesday	Errors: 9	Errors: 21
Wednesday	Errors: 7	Errors: 11
Thursday	Errors: 12	Errors: 18
Total errors	46	99

The above table illustrates the results from week four. During the first day of the week, Monday, the total amount of errors amounted to 18 for Flow pair programming without any time limit, while the amount of errors amounted to 49 for pair programming based on a timer. For Tuesday, the amount of total errors amounted to 9, while timed pair programming had a total number of errors amounting to 21. For Wednesday, the total number of errors for flow pair programming amounted to 7, while the total number of errors for timed pair programming amounted to 11. For the last day of coding, Thursday, the amount of errors amounted to 12 for flow pair programming, while timed pair programming had 18 errors. The amount of error for pair programming based on flow resulted in a total of 46 errors for week 4, while timed pair programming had a result of 99 errors for week 4.

The next evaluation criteria was the amount of time required for code to be written. This evaluation criteria refers to how many qualitative lines of code were written and how fast, and in whole how fast different parts of the software were written.

Table 5: illustrates results week 1

Results week 1	Flow	Time	Individual
Monday	Lines of code: 37	Lines of code: 19	Lines of code: 7
Tuesday	Lines of code: 34	Lines of code: 26	Lines of code: 12
Wednesday	Lines of code: 66	Lines of code: 53	Lines of code: 5
Thursday	Lines of code: 53	Lines of code: 38	Lines of code: 9
Total lines of code	190	136	33

The above table illustrates the results from week one based on the criteria lines of code written. During the first day of the week, Monday, the total lines of code written amounted to

37 for Flow pair programming without any time limit, while the number of lines written amounted to 19 for pair programming based on a timer. In regards to individual programming, 7 lines of code were written. For Tuesday, the number of lines written were 34, while timed pair programming had a total number of lines written of 26. Individual programming provided 12 lines of code that day. For Wednesday, the number of lines written for flow pair programming amounted to 66, while the number of lines written for timed pair programming amounted to 53. The individual programming provided with 5 lines of code. For the last day of coding, Thursday, the number of lines written amounted to 53 for flow pair programming, while timed pair programming had 38 lines of code written and individual programming had 9 lines of code. The number of lines written for pair programming based on flow resulted in a total of 190 errors for week 1, while timed pair programming had a result of 136 lines of code written for week 1. Individual programming had a result of 33 lines of code during the entirety of week 1.

Table 6: illustrates lines of code written in week 2

Results week 2	Flow	Time	Individual
Monday	Lines of code: 24	Lines of code: 43	Lines of code: 10
Tuesday	Lines of code: 33	Lines of code: 25	Lines of code: 11
Wednesday	Lines of code: 70	Lines of code: 15	Lines of code: 29
Thursday	Lines of code: 20	Lines of code: 49	Lines of code: 6
Total lines of code	147	132	56

The above table illustrates the results from week two based on the criteria lines of code written. During the first day of the week, Monday, the total lines of code written amounted to 24 for Flow pair programming without any time limit, while the number of lines written amounted to 43 for pair programming based on a timer. For Tuesday, the number of lines written were 33, while timed pair programming had a total number of lines written of 25. For Wednesday, the number of lines written for flow pair programming amounted to 70, while the number of lines written for timed pair programming amounted to 15. For the last day of coding, Thursday, the number of lines written amounted to 20 for flow pair programming, while timed pair programming had 49 lines of code written. The number of lines written for pair programming based on flow resulted in a total of 147 errors for week 2, while timed pair programming had a result of 132 lines of code written for week 2. The total lines of code written using individual programming during week 2 equals to 56.

Table 7: illustrates lines of code written in week 3

Results week 3	Flow	Time	Individual
Monday	Lines of code: 40	Lines of code: 21	Lines of code: 8
Tuesday	Lines of code: 23	Lines of code: 44	Lines of code: 12
Wednesday	Lines of code: 33	Lines of code: 40	Lines of code: 24
Thursday	Lines of code: 47	Lines of code: 25	Lines of code: 16
Total lines of code	143	130	60

The above table illustrates the results from week three based on the criteria lines of code written. During the first day of the week, Monday, the total lines of code written amounted to 40 for Flow pair programming without any time limit, while the number of lines written amounted to 44 for pair programming based on a timer. For Tuesday, the number of lines written were 23, while timed pair programming had a total number of lines written of 44. For Wednesday, the number of lines written for flow pair programming amounted to 33, while the number of lines written for timed pair programming amounted to 40. For the last day of coding, Thursday, the number of lines written amounted to 47 for flow pair programming, while timed pair programming had 25 lines of code written. The number of lines written for pair programming based on flow resulted in a total of 143 errors in the third week, while timed pair programming had 130 lines of code written during week 3. The total lines of code written using individual programming equals to 60 in week 3.

Table 8: illustrates lines of code written in week 4

Results week 4	Flow	Time	Individual
Monday	Lines of code: 50	Lines of code: 34	Lines of code: 23
Tuesday	Lines of code: 73	Lines of code: 44	Lines of code: 15
Wednesday	Lines of code: 38	Lines of code: 49	Lines of code: 21
Thursday	Lines of code: 47	Lines of code: 30	Lines of code: 17
Total lines of code	207	157	76

Table 8 displays the results of week 4 based on the criteria lines of code written. During the first day of the week, Monday, the total lines of code written amounted to 50 for Flow pair programming, while the number of lines written was 34 for pair programming based on a timer. During Tuesday 73 lines were written when using Flow pair programming, while timed pair programming had 44 lines of code written. For Wednesday, the number of lines written for flow pair programming amounted to 38, while the number of lines written for timed pair programming amounted to 49. For the last day of coding, Thursday, the number of lines

written amounted to 47 for flow pair programming, while timed pair programming had 30 lines of code written. The number of lines written for pair programming based on flow resulted in a total of 207 errors in the fourth week, while timed pair programming had 157 lines of code written during week 4. The total lines of code written using individual programming equals to 76 in week 4.

The final evaluation criteria was how fast problems were solved. The results here were measured according to a scale of slow-medium-fast. This criteria was based upon individual programming, flow pair programming and timed pair programming.

Table 9: illustrates how fast problems were solved in week 1

Results week 1	Pair Programming	Individual Programming
Monday	medium	slow
Tuesday	fast	medium
Wednesday	medium	slow
Thursday	medium	slow
Average	medium	slow

The above table illustrates the results from week one based on the criteria of how long it took to solve a problem. Here the two different methods that were compared were pair programming in general and individual programming. For Monday, the perceived quickness of task and problem solving was medium, while Individual programming had a perceived quickness of slow. For Tuesday, the perceived quickness of task and problem solving was medium, while Individual programming had a perceived quickness of slow. For Wednesday, the perceived quickness of task and problem solving was medium, while Individual programming had a perceived quickness of slow. For Thursday, the perceived quickness of task and problem solving was medium, while Individual programming had a perceived quickness of slow. The average perceived quickness of pair programming for week one was medium, while the average perceived quickness of individual programming for week one was slow.

Table 10: illustrates how dast problems were solved in week 2

Results week 2	Pair Programming	Individual Programming
Monday	medium	medium
Tuesday	medium	medium
Wednesday	fast	medium
Thursday	medium	slow
Average	medium	medium

Table 10 shows the results from week two based on the criteria of how long it took to solve a problem. Here the two different methods that were compared were pair programming in general and individual programming. For Monday, the perceived quickness of task and problem solving was medium for pair programming which is similar to the result of individual programming. Tuesday had the exact same results as Monday. For Wednesday, the perceived quickness of task and problem solving was fast for pair programming, while Individual programming had a perceived quickness of medium. For Thursday, the perceived quickness of task and problem solving was medium for pair programming, while Individual programming had a perceived quickness of slow. The average perceived quickness of pair programming for week two was medium, while the average perceived quickness of individual programming for week two was slow.

Table 11: illustrates how dast problems were solved in week 3

Results week 3	Pair Programming	Individual Programming
Monday	medium	fast
Tuesday	fast	medium
Wednesday	fast	fast
Thursday	fast	medium
average	medium-fast	medium

Table 11 shows the results from week three based on the criteria of how long it took to solve a problem. Here the two different methods that were compared were pair programming in general and individual programming. For Monday, the perceived quickness of task and problem solving was medium for pair programming which differs to the result of individual programming which was fast. Tuesday was fast for pair programming while the individual programming scored medium. For Wednesday, both pair- and individual programming scored fast as results. For Thursday, the perceived quickness of task and problem solving was fast for pair programming, while individual programming had a perceived quickness of medium. The average perceived quickness of pair programming for week three was medium-fast, while the average perceived quickness of individual programming for week two was medium.

Table 12: illustrates how dast problems were solved in week 4

Results week 4	Pair Programming	Individual Programming
Monday	medium	medium
Tuesday	fast	medium
Wednesday	medium	slow
Thursday	fast	fast
average	medium - fast	medium

Table 12 provides the results from week four based on the criteria of how long it took to solve a problem. Here the two different methods that were compared were pair programming in general and individual programming. For Monday, the perceived quickness of task and problem solving was medium for both pair programming and individual programming. During Tuesday pair programming scored fast while the individual programming scored medium. For Wednesday, pair programming scored medium while individual programming scored slow. For Thursday, the perceived quickness of task and problem solving was fast for both pair programming and individual programming. The average perceived quickness of pair programming for week four was medium-fast, for both of the programming methods.

Section V: Analyze your results and evaluate your project work

The first evaluation criteria to be analyzed is Correctness. When evaluating the data in Section IV, it is very clear that programming based on flow is a much more effective method. To be able to program based on flow, we believe that the two programmers must have had either experience working with each other or known each other prior on a friendly basis. This is because there needs to be an established ground of respect and trust between the two programmers programming based on flow.

Let's say that, for example, two unfamiliar people are given the task to pair-program based on flow. These two individuals don't know eachothers skill levels and they have never worked together before and in addition to this they don't know each other's personality types. What one person might deem to be a reasonable amount of time spent programming might be offensive to the other person. Therefore, many conflicts may arise between unfamiliar people who pair-program based on flow. In that case, it might be better for these two people to start off with timed programming, and after getting to know each other they could get into the habit of programming on flow.

The anecdotal evidence during these four weeks points to flow based pair programming being superior to timed programming. It should be taken into account that the two authors of this report have known each other and studied together daily for about two years, and have grown accustomed to each other's programming styles. The importance of building a relation before starting to pair-program based on flow is also pointed out in the paper 'The Costs and Benefits of Pair Programming', (Cockburn & Williams).[1]

Furthermore, the difference between pair programming overall and individual programming is the amount of errors accumulated. A study on 40 senior Computer Science students concluded that students who pair programmed had solutions with 15% less bugs compared to students who programmed individually on the same assignment (Williams, 2000).

When looking at tables 1-4 in section IV, you can see that sometimes that total amount of errors per week were double when timed pair programming was used. This could be a bit misleading, as even though programming based on flow was more beneficial, the amount of errors after each switch were due to there being a specified time limit before each change.

---

[1] http://www.diva-portal.org/smash/get/diva2:832682/FULLTEXT01.pdf

This would lead to one developer having to switch seats when in the middle of writing a piece of code, which in some cases resulted in a few errors. Meanwhile, when programming on flow, the team member doing the physical coding would give up the seat when either stuck or when done with a specific part, which in turn would only lead to errors accumulating when a member was stuck on a specific part.

The second evaluation criteria to be analyzed is the amount of time required for code to be written. When studying the data in Section IV it appears that throughout the entirety of the four weeks, the flow version of pair programming was dominant in the matter of total number of lines code written. Table 5 illustrates the results of week 1 where the total lines of code written using flow was 190 meanwhile the total lines of code written using timed based pair programming was 136. Here it is visible that there is a ratio of 1.4 between them, meaning that flow- based pair programming during this week resulted to be 40 percent more effective solely based on the number of codes written. Furthermore, studying Table 6 which shows the results of week two, the ratio between flow based and timed based pair programming appears to be 1.1. This figure strengthens that flow based programming has been 10% more effective this week. Table 7 illustrates the results of week 3 where total lines of code written using flow was 143 while time was 130. This gives a ratio of exactly 1.1 between them. This ratio is in line with last week's results. Moreover, Table 8 illustrates the results of the last week. The total number of lines written using flow was 207 meanwhile the timed based pair programming resulted in 157 lines written. This gives a ratio of 1.3 meaning that flow was 30% more effective than time in relation to the amount of lines of codes written.

Overall, our study shows that flow was a more effective way of doing pair programming when looking at this particular criteria due to the fact that we managed to write more lines of code in the same amount of time. However, the results indicate that there is a devastating difference between pair programming and individual programming in terms of total lines of code written in a specific space of time. In Table 5, the total lines of code written by individual programming is visible for week 1 which is 33 lines. This in comparison to both flow- based pair programming and time- based pair programming displays a massive difference between them. The same is proven in Table 6 where the total lines of code using individual programming equals to 56 lines. The same table illustrates that the total lines of code using flow- based pair programming is equal to 147 during the same week while time- based pair programming is equal to 132. Both of the pair programming methods resulted in more than the double amount of lines written in individual programming during week 2. Furthermore, when studying week 3 and week 4 that is visible in Table 7 and Table 8 the results are very much alike and proves the same thing.

In conclusion, our study shows that pair programming is more effective than individual programming based on how fast code is written. An experiment about how fast problems are solved by students working in pairs and individuals is described in an article. The results of this experiment shows that the people working in pairs needed 15% more time to code and solve the different problems (Cockburn & Williams). This differs from our study that instead shows that coding is faster when pair programming. The difference in result could depend on many different factors, a fact that might be worthy to consider is how well the two people programming together know each other. In our study both programmers know each other very well which we would consider very helpful when programming together.

The third and last evaluation criteria was how fast problems were solved. Here there was no comparison between timed programming and programming based on flow, as the amount of dialogue between the two group members was the same. Instead, the comparison was solely made between pair programming in general and individual programming. The results from tables 9-12 in section IV tend to favor pair programming.

Looking at table 9, we see that pair programming was more effective when solving problems. The reason for this is mainly that there were many new languages and methods to be learnt in the beginning of the project, which in turn was easier to learn by having an open dialogue and sharing skills with each other than sitting by yourself and trying to figure it out alone.

However, during week 2 of the project, the results were pretty similar when comparing pair programming and individual programming. Table 10 in section IV illustrates that there only was one day that week where problem solving was easier with pair programming, while the rest of the days had an equal quickness in how to solve the programs. This is probably due to being more familiar with the topics covered during the project.

Furthermore, week 3 and week 4 consisted of similar results to week 1. Here individual programming was enough to solve problems, but solving tasks in pairs was proven to be much more effective than the solo route. During week 4, when many aspects of the code had to be fine tuned, it really did help to be able to have a dialogue with your pair programming partner about all the issues at hand. When sitting alone with a problem we tend to get tunnel vision and not be able to think outside of the problem scope. This was however not the case when pair programming, as many ideas were presented and we could argue and have a dialogue about what works the best.

Lastly, findings in through international studies prove our above statements about the effectiveness of pair programming when it comes to problem solving, as it shows that pairs of programmers outperformed their individual counterparts and also enjoyed the process of problem-solving more and were also more confident in their solutions. The statistics of this study point to a 40% increase in task completion when programming in pairs rather than doing it individually (Nosek, 1998).

SECTION VI: Conclusions and Future Work

During these four weeks a variety of different programming methods were trialed, tested and compared to international findings. The effectiveness of pair programming overall was put up to the test against individual programming. The results were in every aspect in favor of pair programming, and many international articles that were referred to agreed with the conclusion. Furthermore, after a review of countless different pair programming methods, two methods were carefully selected. The selection was made according to personal experience with pair programming and international findings in the specific area. One of the two pair programming methods used was Timed pair programming. It worked better than individual programming and is a proven method. The other method was flow based pair programming. This was the method that yielded the best results between the two types of pair programming. The reason for this is believed to be that it works better when two

programmers are already familiar with each other's programming styles, which was the case during this project.

References

Alistair Cockburn, Laurie Williams. The Costs and Benefits of Pair Programming. Retrieved on May 27, 2022, from https://bzupages.com/attachments/3307d1232491412-costs-benefits-pair-programming.pdf

Jan Chong, Tom Hurlbutt. (2007). The Social Dynamics of Pair Programming, Retrieved on May 27, 2022, from https://ieeexplore.ieee.org/abstract/document/4222597/references#references

Cprime. (2022). WHAT IS AGILE? WHAT IS SCRUM? Retrieved on May 27, 2022, from https://www.cprime.com/resources/what-is-agile-what-is-scrum/

Nosek, J.T. (1998). The Case for Collaborative Programming. Retrieved on May 27, 2022, from https://search.library.ucsb.edu/discovery/openurl?institution=01UCSB_INST&rfr_id=info:sid%252Fprimo.exlibrisgroup.com-bX-Bx&rfr_id=info:sid%2Fprimo.exlibrisgroup.com-169946-Bx&rft_val_fmt=info:ofi%2Ffmt:kev:mtx:&rft.epage=108&rft.volume=41&rft_id=info:doi%2F&resource_type=article&rft.isbn_list=&rft.jtitle=Communications%20of%20the%20ACM&rft.genre=article&rft.issue=3&rft.auinit1=J&rft.aulast=Nosek&rft.auinit=J.T.&rft.date=1998&rft.eisbn_list=&rft.spage=105&rft.au=Nosek,%20J.T.%20T&rft.atitle=The%20case%20for%20collaborative%20programming&rft.issn=0001-0782&rft.eissn=1557-7317&svc_dat=CTO&vid=01UCSB_INST:UCSB.

Denisse Morelos. (2018). Traditional vs. Agile Software Development Method: Which One is Right for Your Project? Retrieved on May 27, 2022, from https://dzone.com/articles/traditional-vs-agile-software-development-method-w

Indusree Mavuru. (2018). Traditional vs. Agile Software Development Methodologies. Retrieved on May 27, 2022, from https://www.kpipartners.com/blog/traditional-vs-agile-software-development-methodologies

Alexander S. Gillis. (2021). Pair Programming. Retrieved on May 27, 2022, from https://www.techtarget.com/searchsoftwarequality/definition/Pair-programming

Jo E. Hannay, Tore Dybå, Erik Arisholm, Dag I.K. Sjøberg. (2009). The effectiveness of pair programming: A meta-analysis. Retrieved on May 27, 2022, from https://www.sciencedirect.com/science/article/abs/pii/S0950584909000123

(Cockburn & Williams), The Costs and Benefits of Pair Programming, page. 4, Retrieved on May 27, 2022, from https://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF

Wiliams, L. A. (2000), The Collaborative Software Process, Retrieved on May 27, 2022, from https://dl.acm.org/doi/10.5555/931413

Ghassan Alkadi, 285. Software Engineering I. Retrieved on May 28, 2022, from https://www2.southeastern.edu/Academics/Faculty/galkadi/285/notes/Chapter5.doc